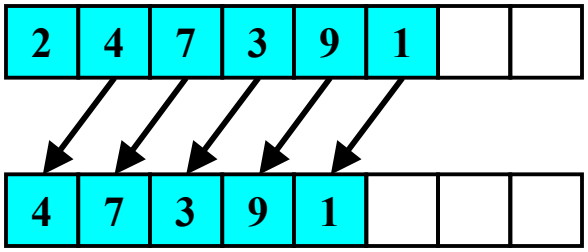
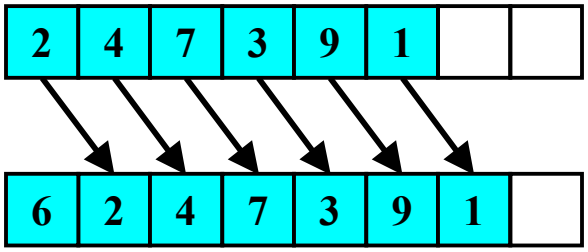
Worksheet 20: Solution: Dynamic Array Deque and Queue

In an earlier lesson you explored why one cannot use a **dynamic array** as the basis for an efficient queue (which adds to the back and removes from the front) or deque (allowing for addition and removal from either end). This is because adding to or removing from the first location (that is, index position 0) is very slow O(*n*).

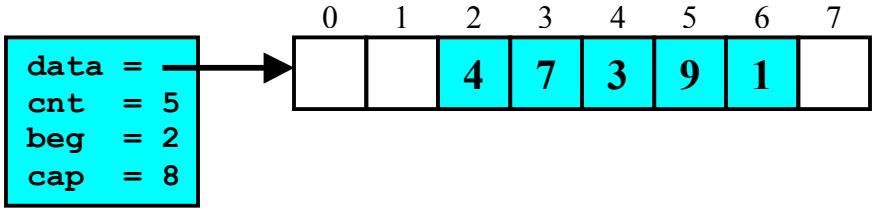
Removing a value requires elements to slide left.



Adding a value requires elements to slide right.



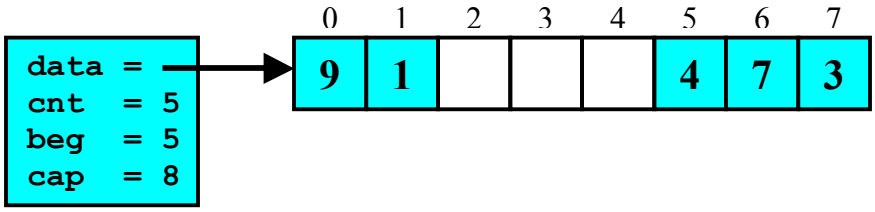
This does not mean that a dynamic array-*like* data structure cannot be used to implement a queue. The key insight is that we can allow the starting location of the block of elements to "float", rather than being fixed at location zero. An internal integer data field (**`beg`**) records the current beginning or starting location.



Notice that the "logical" index no longer corresponds to the physical index. The value with logical index zero is found in this diagram at array location 2. The value with logical index 1 is found at location 3, and so on.

With this change, it is now easy to implement additions or removal from either front or back. To add to the front, simply decrement the starting location, and place the new element in the new starting place. To add to the back, simply increase the size, and place

the new value in the location determined by the addition of the starting location and size. But there is one subtle complexity. The block of values stored in the collection can wrap



around from the end back to the beginning:

Here the block of elements begins at index position 5. The next two elements are found in index positions 6 and 7. But the element after that is found at index position zero. As with the dynamic array, the internal array must be reallocated and doubled if the count of elements becomes equal to the array capacity.

Using this idea, complete the implementation of the deque. Implement the methods that will add, retrieve, or remove an element from either the front or back. Explain why each operation will have constant (or amortized constant) execution time. Make sure you write **`_doubleCapArrDeque`**.

`See posted code: dynamicArrayDeque.c`