Jacob Karcz

# Lab Fa

Answers to questions on Lab sheet:

We only need a singly linked list to implement a stack because a stack only ever deals with the "head" of the list. The last node added (or the last one that has not been erased) is the only one any of the stack member functions care to deal with. Also, because we are erasing nodes as we pop items off the stack, there is no need to keep track of any empty values.

The second question is more thoroughly answered in the output of my initial driver program, found after my answers. But in short, you can tell if they are working correctly because a stack retrieves items in the opposite order than they were entered (first-in, last-out). Stacks retrieve the most recent data. On the other hand, the queue retrieves items in the same order they were entered (first-in, first-out).

Lastly, assuming the stack and queue is implemented correctly, only one item is accessible at a time. Thus if you choose to print (or retrieve) all items entered plus one you will get a null variable for the last item (since the structure is empty. Alternatively, the isEmpty() function can be called resulting in an error message on the last item. As long as we are dealing with dynamic stacks and queues, we don't need to a test for an upper limit of entries because these are dynamic data structures. We can add as many items as we want until the system runs out of memory. There is no upper limit with dynamic stacks and queues. However, if it was a static stack or queue, because they are commonly based off arrays there would need to be a message that the limit was reached.

**Dynamic Stack and Queue Driver Program (labFaAutoDriver.cpp) Output:**

**Lets begin by implementing a dynamic stack and pushing 5 values:**

```
Push: 5
Push: 10
Push: 15
Push: 20
Push: 25

Pop: 25
Pop: 20
Pop: 15
Pop: 10
Pop: 5
```

**The stack is a first-in last-out data structure. As we can see, 5, the first number entered, is the last one to be retrieved.**
**The data in the stack is accessible in reverse relative to the order in which it was added.**

**Now lets implement a dynamic queue and enqueue the same 5 numbers:**

```
Enqueue: 5
Enqueue: 10
Enqueue: 15
Enqueue: 20
Enqueue: 25

Dequeue: 5
Dequeue: 10
Dequeue: 15
Dequeue: 20
Dequeue: 25
```

**The queue is a FIFO data structure (first-in last-out. As we can see, 5, the first number entered, is the first one to be retrieved.**
**The data in the queue is accessible in the same order in which it was entered.**

**To reiterate,**

```
this is a stack:
Adding: 1 2 3 4 5
Removing: 5 4 3 2 1

and this is a queue:
Adding: 1 2 3 4 5
Removing: 1 2 3 4 5

Program ended with exit code: 0
```