CO322: Data Structure and Algorithms

Lab 01 – Part 2: Queues

February 10, 2016

- ## Queue

Queue is a specialized data storage structure (Abstract data type). Similar to a Stack, access of elements in a Queue is restricted. It has two main operations:

- ***Enqueue*:** Insert a data element in to a Queue
- ***Dequeue*:** Remove a data element from a Queue

An item can be inserted at the end ('rear') of the queue and removed from the front ('front') of the queue. It is therefore, also called First-In-First-Out (FIFO) list.

Queue has the following properties, which may depend on the implementation method:

- **Capacity:** the maximum number of elements the Queue can hold
- **Size:** the current size of the Queue
- **front:** pointer to the first element of the Queue
- **rear:** pointer to the last element of the Queue

As in the case of Stacks, the Data Elements of a Queue are usually represented by an ***Array*** or a ***Linked List***.

- **Array Implementation:**
  An Array implementation of Queue is much more complicated than that of a Stack, since the Queue implementation has to keep track of the ***front*** and the ***rear*** of the Queue, both of which could (& would) change as data elements are inserted & removed.

  Since the data elements are added from one end (***rear***), while those are removed from the other (***front***), an Array implementation of Queue could (& should) use a *Circular Array*. This complicates it even further as the *rear* could wrap-around, move up and catch the *front*, filling up the entire queue. Alternatively, the *rear* pointer being *equal* to the *front* pointer could indicate that the *front* has caught up to *rear*, at which point the Queue would be empty.

  Thus, you need to clearly define the *FULL* and *EMPTY* instances of a Queue implemented using a Circular Array. Moreover, you need to handle increments to both *front* & *rear* beyond the capacity of the array (i.e., wrap-around).

- **Linked-List Implementation:**
  On the other hand, a Linked-List implementation of a Queue is much simpler, as tracking the *front* & the *rear* of the queue using data pointers is sufficient to provide the FIFO functionality.

  The capacity of the queue is usually not defined in a linked-list implementation, while the size could either be calculated or tracked using an additional variable.

Exercise 1

1.1. Implement the Queue ADT using a Circular Array to store the data elements. Your implementation should have the following functions:

- *queueCreate* (create the Queue Data Structure)
- *queueDestroy* (destroy/clean Queue Data Structure)
- *enqueue*
- *dequeue*
- *queuePeek* (peek at the top data element of the Queue – does not dequeue)
- *queueIsEmpty* (check whether the Queue is empty)
- *queueIsFull* (check whether the Queue is full – no space in the Array!)

Bonus marks: write a program to demonstrate the functionality of your Queue implementation.

Exercise 2

2.1. Implement the Queue ADT using a Linked List to store the data elements. Your implementation should have the following functions:

- *queueCreate* (create the Queue Data Structure)
- *queueDestroy* (destroy/clean Queue Data Structure)
- *enqueue*
- *dequeue*
- *queuePeek* (peek at the top data element of the queue – does not dequeue)
- *queueIsEmpty* (check whether the queue is empty)