ICS 2105:
Data Structures and Algorithms

Queue Data Structure

Acknowledgement

- Notes adapted from Data structures Using C++ by Malik, studytonight.com and tutorials point.

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Define queue data structure.
  - Describe queue operations.
  - Implement a queue data structure using C++.

Introduction

- The notion of a queue in computer science is the same as the notion of the queues to which you are accustomed in everyday life.
- There are queues of:
  - Customers in a bank or in a grocery store and
  - Cars waiting to pass through a tollbooth.

Introduction

- Similarly, because a computer can send a print request faster than a printer can print, a queue of documents is often waiting to be printed at a printer.

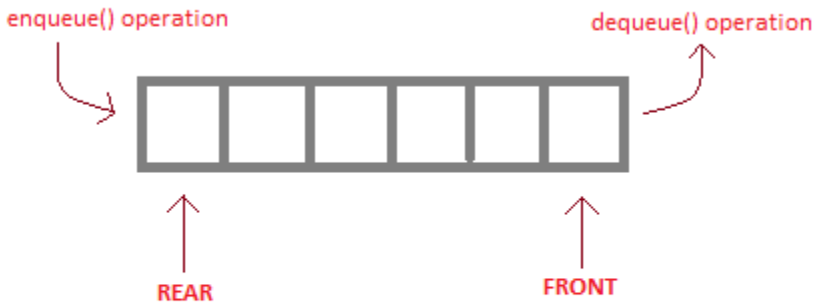
Queues

- The general rule to process elements in a queue is that the customer at the front of the queue is served next and that when a new customer arrives, he/she stands at the end of the queue.
  - That is, a queue is a First In First Out data structure.

What is a Queue?

- A data structure in which the elements are added at one end, called the rear, and deleted from the other end, called the front; a First In First Out (FIFO) data structure.

Queue



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Queue Representation

Queue

- A queue is a set of elements of the same type in which the elements are added at one end, called the back or rear, and deleted from the other end, called the front.
  - E.g., consider a line of customers in a bank, wherein the customers are waiting to withdraw/deposit money or to conduct some other business.
  - Each new customer gets in the line at the rear.
  - Whenever a teller is ready for a new customer, the customer at the front of the line is served.

Queue

- The rear of the queue is accessed whenever a new element is added to the queue, and the front of the queue is accessed whenever an element is deleted from the queue.
- As in a stack, the middle elements of the queue are inaccessible, even if the queue elements are stored in an array.

Queue Implementation

- As in stacks, a queue can also be implemented using:
  - Arrays
  - Linked-lists
  - Pointers and Classes.
- For the sake of simplicity, we shall implement queues using one-dimensional array

Queue Operations

- From the definition of queues, we see that the two key operations are:
  - Add and
  - Delete.
- We call the add operation addQueue and the delete operation deleteQueue.

Queue Operations

- Because elements can be neither deleted from an empty queue nor added to a full queue, we need two more operations to successfully implement the addQueue and deleteQueue operations:
  - isEmptyQueue (checks whether the queue is empty) and
  - isFullQueue (checks whether a queue is full).

Queue Operations

- We also need an operation, initializeQueue, to initialize the queue to an empty state.
- Moreover, to retrieve the first and last elements of the queue, we include the operations front and back.

Queue Operations

- initializeQueue:
  - Initializes the queue to an empty state.
- isEmptyQueue:
  - Determines whether the queue is empty. If the queue is empty, it returns the value true; otherwise, it returns the value false

Queue Operations

- isFullQueue:
  - Determines whether the queue is full. If the queue is full, it returns the value true; otherwise, it returns the value false.
- Front:
  - Returns the front, that is, the first element of the queue. Prior to this operation, the queue must exist and must not be empty.

Queue Operations

- Back:
  - Returns the last element of the queue. Prior to this operation, the queue must exist and must not be empty.
- addQueue:
  - Adds a new element to the rear of the queue. Prior to this operation, the queue must exist and must not be full.

Queue Operations

- deleteQueue:
  - Removes the front element from the queue. Prior to this operation, the queue must exist and must not be empty.

Applications of Queue

- Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
- Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
- Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

Implementation of Queue Data Structure

- Initially the head(FRONT) and the tail(REAR) of the queue points at the first index of the array (starting the index of array from 0).
- As elements are added to the queue, the tail keeps on moving ahead, always pointing to the position where the next element will be inserted, while the head remains at the first index.

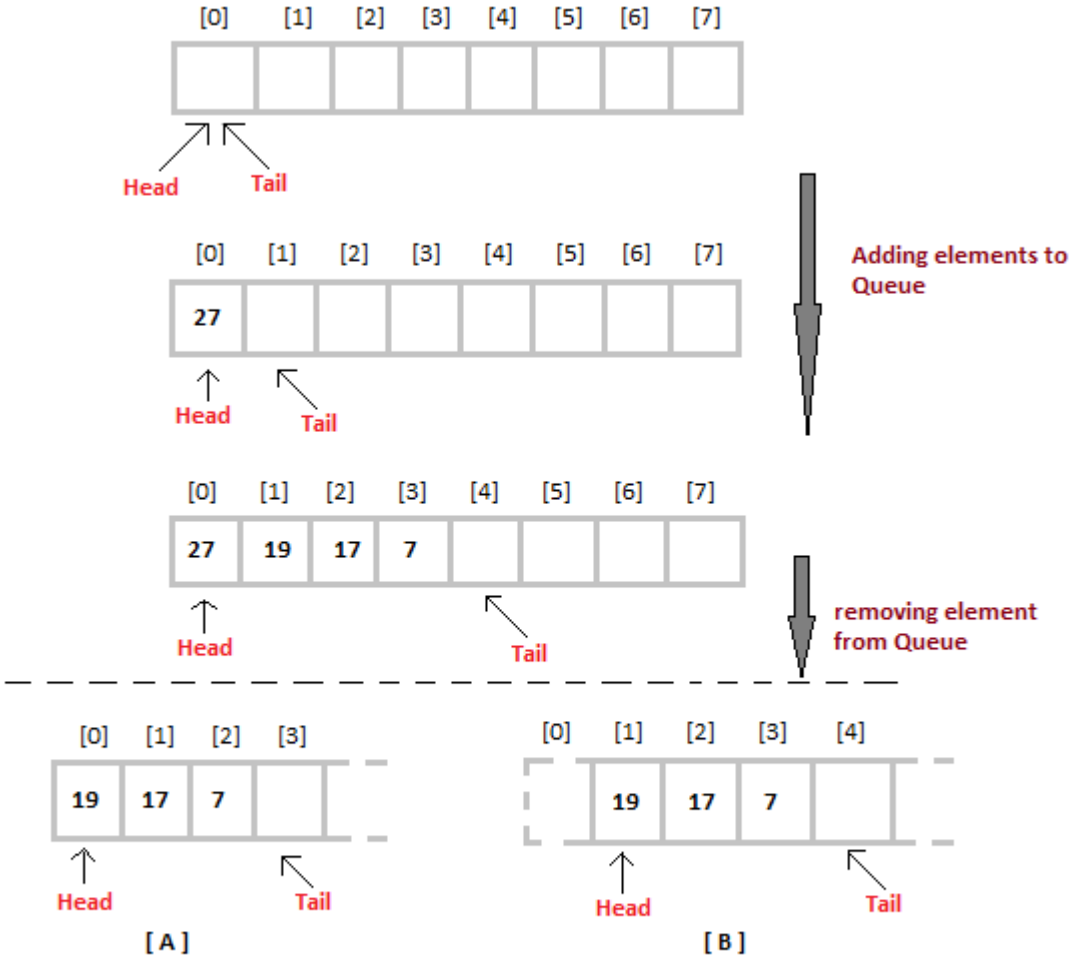
Implementation of Queue Data Structure

- When we remove an element from Queue, we can follow two possible approaches:
  - Remove the element at head position, and then one by one shift all the other elements in forward position
  - Remove the element from **head** position and then move **head** to the next position.

Implementation of Queue Data Structure

- In the first approach, there is an **overhead of shifting the elements one position forward** every time we remove the first element.
- In the second approach, there is no such overhead, but whenever we move head one position ahead, after removal of first element, the **size on Queue is reduced by one space** each time.

Implementation of Queue Data Structure



Algorithm for ENQUEUE operation

1. Check if the queue is full or not.
2. If the queue is full, then print overflow error and exit the program.
3. If the queue is not full, then increment the tail and add the element.

Algorithm for DEQUEUE operation

1. Check if the queue is empty or not.
2. If the queue is empty, then print underflow error and exit the program.
3. If the queue is not empty, then print the element at the head and increment the head.

End of lesson