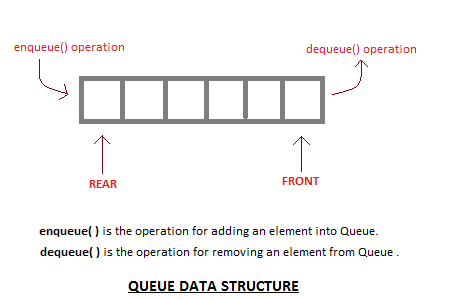
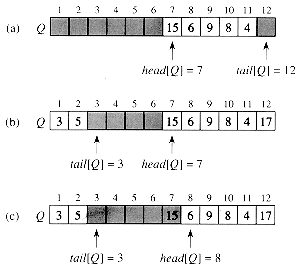
**Queue data structure**

A **queue** is a data structure that contains a collection of elements. The queue implements a first-in, first out or **FIFO** mechanism (policy), i.e. the element that is inserted first is also deleted first.

There are several efficient ways to implement queues on a computer. Consider how to use a simple array to implement a queue.

The INSERT operation on a queue is called **ENQUEUE**, and the DELETE operation on a queue is called **DEQUEUE**. Like the stack operation POP, DEQUEUE takes no element argument. The queue has a **head** and a **tail.** When an element is enqueued, it takes its place at the tail of the queue. The element dequeued is always the one at the head of the queue.





**Figure. A queue implemented using an array Q[1 . . 12]. Queue elements appear only in the lightly shaded positions. (a) The queue has 5 elements, in locations Q [7..11]. (b) The configuration of the queue after the calls ENQUEUE(Q, 17), ENQUEUE(Q, 3), and ENQUEUE(Q, 5). (c) The configuration of the queue after the call DEQUEUE(Q) returns the key value 15 formerly at the head of the queue. The new head has key 6.**

This Figure shows one way to implement a queue of at most n - 1 elements using an array Q [1..n]. The queue has an attribute head [Q] that indexes, or points to, its head. The attribute tail [Q] indexes the next location at which a newly arriving element will be inserted into the queue. The elements in the queue are in locations head[Q], head [Q] + 1, . . . , tail [Q] - 1, where we "wrap around" in the sense that location 1 immediately follows location n in a circular order. When head [Q] = tail [Q], the queue is empty. Initially, we have head [Q] = tail [Q] = 1. When the **queue is empty**, an attempt to dequeue an element causes the queue to **underflow.** When head [Q] = tail [Q] + 1, the **queue is full**, and an attempt to enqueue an element causes the queue to **overflow.**

The following program fragment performs ENQUEUE operation of the queue.

ENQUEUE(Q, x)

1. Q [tail [Q]] ← x

2. if tail [Q] = length [Q]

3. then tail [Q] ← 1

4. else tail [Q] ← tail [Q] + 1

The following program fragment performs DEQUEUE operation of the queue.

DEQUEUE(Q)

1. x ← Q [head [Q]]

2. if head [Q] = length [Q]

3. then head [Q] ← 1

4. else head [Q] ← head [Q] + 1

5. return x

**Applications of queue data structures**

[Queue](http://en.wikipedia.org/wiki/Queue_%28data_structure%29)is used when things don’t have to be processed immediately, but have to be processed in  **F**irst **I**n **F**irst **O**ut order

1. Operating systems schedule jobs (with equal priority) in the order of arrival. Waiting lists.
2. Access to shared resources (When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling, printer)
3. Simulations (Simulation of real-world queues such as lines at a ticket counter or any other first-come first-served scenario requires a queue.)
4. Auxiliary data structure for algorithms (queue, priority queue). Component of other data structures.