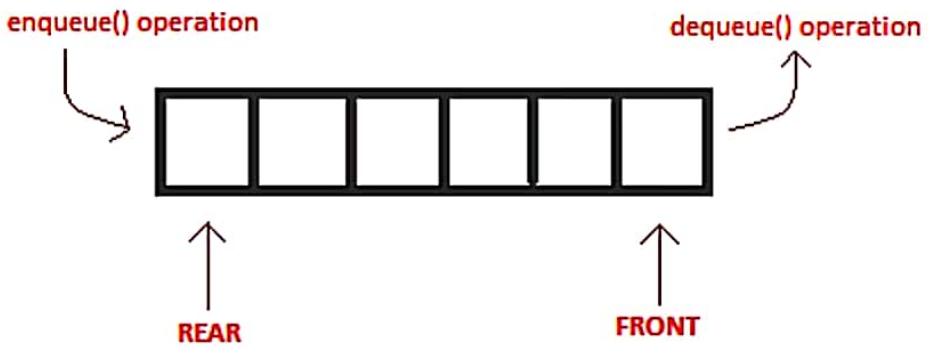
# Queue

**Queue** is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the **REAR**(also called **rear**), and the removal of existing element takes place from the other end called as **FRONT**(also called **front**).

This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.

The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.




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

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

**QUEUE DATA STRUCTURE**

## Applications of Queue

Queue 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 like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.

2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

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

### Basic features of Queue

- Like stack, queue is also an ordered list of elements of similar data types.
- Queue is a FIFO( First in First Out ) structure.
- Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
- peek( ) function is oftenly used to return the value of first element without dequeuing it.

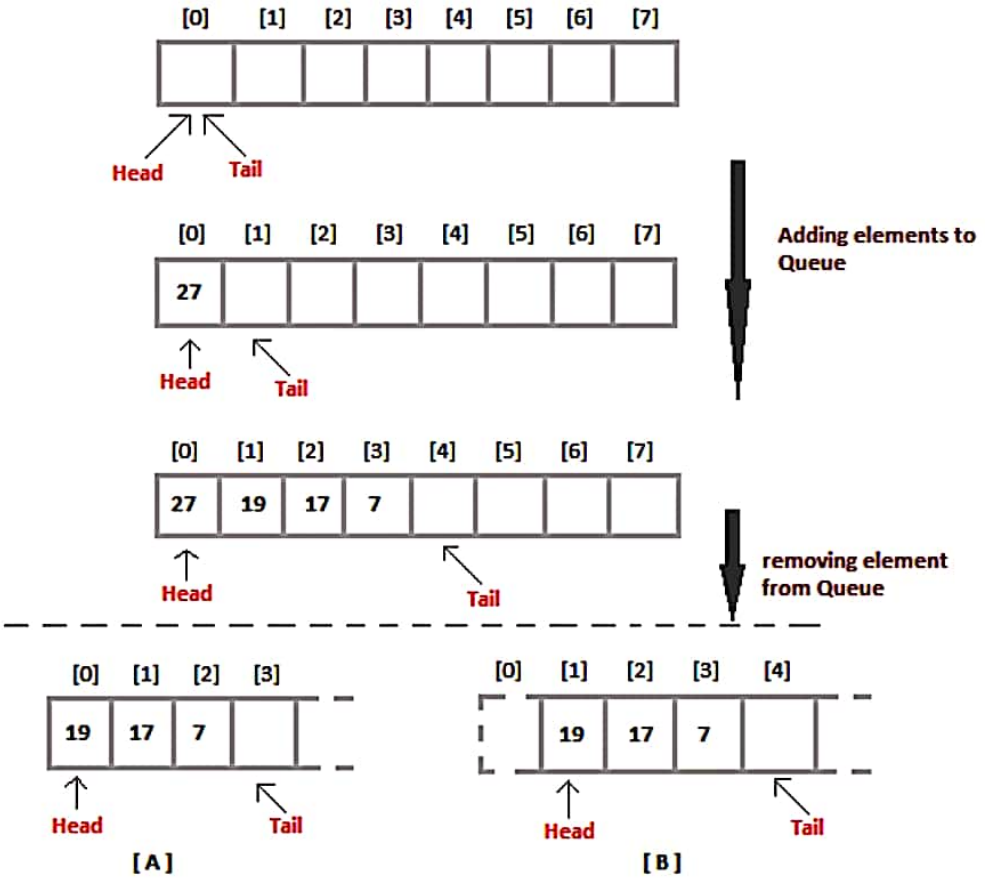
## Types of Queue:

1. Linear Queue
2. Circular Queue
3. Doubled Ended Queue (D-Queue)
4. Priority Queue

### Implementation of Queue Data Structure

Queue can be implemented using an Array, Stack or Linked List. The easiest way of implementing a queue is by using an Array.

Initially the front (FRONT) and the rear (REAR) of the queue points at the first index of the array (starting the index of array from 0). As we add elements to the queue, the rear keeps on moving afront, always pointing to the position where the next element will be inserted, while the front remains at the first index.




When we remove an element from Queue, we can follow two possible approaches (mentioned [A] and [B] in above diagram). In [A] approach, we remove the element at front position, and then one by one shift all the other elements in forward position.

In approach [B] we remove the element from front position and then move front to the next position.

In approach [A] there is an overfront of shifting the elements one position forward every time we remove the first element.

In approach [B] there is no such overfront, but whenever we move front one position afront, after removal of first element, the size on Queue is reduced by one space each time.

**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 rear 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 front and increment the front.

## Linear Queue Implementation

# Insertion

**Q insertion(Q, max, item, front, rear)**

**Step1: If (rear==max-1) then**

**Print "overflow"**

**Step2: Else**

**if(rear==-1 and front==-1)**

**set front=0**

**rear = 0**

**Step 3: Else**

**rear=rear+1**

**[End of if]**

**Step 4: Q[rear]=item**

**Step 5: Stop**

## Deletion

**Q deletion(Q, max, item, front, rear)**

**Step1: Start**

**Step2: If (rear==-1 and front==-1)**

**Print "underflow"**

**End of if, exit.**

**Step3: item = Q[Front]**

**print "The deleted item is" item**

**Step 4: If (rear==front)**

**set rear =-1**

**front=-1**

**Step 5: else**

**front=front+1**

**[End of if]**

**Step 6: Stop**

## Display

**Q display(Q, max, front, rear)**
**Step1: Start**
**Step2: If (front==-1 and rear==-1 )**
**Print "no element for display"**
**Step3: else**
**Step 3.1: repeat for (i =front to rear by +1)**
**Step 3.1.1: print Q[i]**
**[End of for]**
**[End of if ]**
**Step 6: Stop**