Queue data structure is similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first. A real-world example of a queue can be a single-lane one-way road, ticket counter and bus-stops. Queue is a FIFO( First in First Out ) structure.

## Queue Representation

As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures.

## Basic Operations

* enqueue() − add (store) an item to the queue.
* dequeue() − remove (access) an item from the queue.

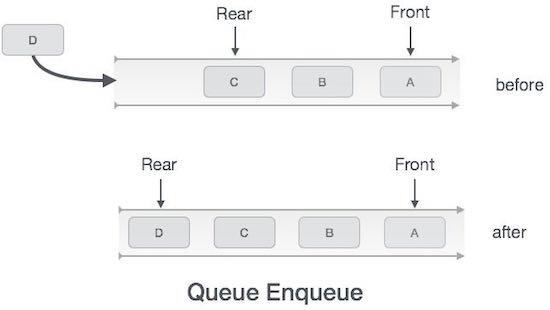
Few more functions are required to make the above-mentioned queue operation efficient. These are −

* peek() − Gets the element at the front of the queue without removing it.
* isfull() − Checks if the queue is full.
* isempty() − Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by the front pointer and while enqueing (or storing) data in the queue we take help from the rear pointer.

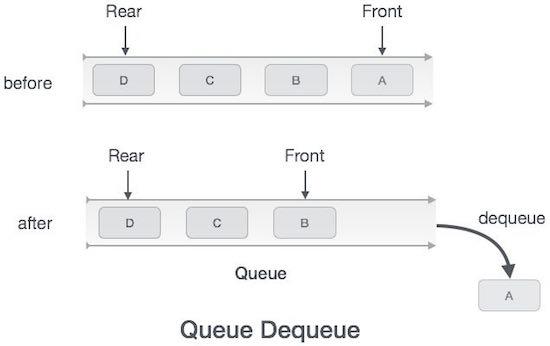
## Enqueue Operation

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.



## 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 head.



### Complexity Analysis of Queue Operations

Just like Stack, in case of a Queue too, we know exactly, on which position new element will be added and from where an element will be removed, hence both these operations requires a single step.

* Enqueue: O(1)
* Dequeue: O(1)
* Size: O(1)

## Complexity

| **Data Structure** | **Time Complexity** | | | | | | | | **Space Compleity** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Average** | | | | **Worst** | | | | **Worst** |
|  | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion |  |
| Queue | θ(n) | θ(n) | θ(1) | θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |

# Types of Queues

### 

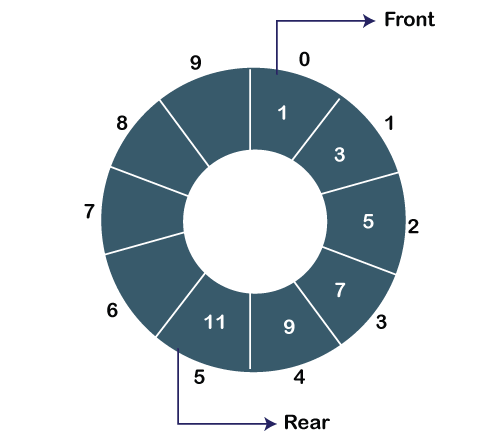
**There are four types of Queues:**

* **Linear Queue**

In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end. It strictly follows the FIFO rule.

* **Circular Queue**

In the Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as **Ring Buffer** as all the ends are connected to another end. The circular queue can be represented as:



* **Priority Queue**

A priority queue is another special type of Queue data structure in which each element has some priority associated with it. Based on the priority of the element, the elements are arranged in a priority queue. If the elements occur with the same priority, then they are served according to the FIFO principle.

# Double Ended Queue

Double ended queue is a more generalized form of queue data structure which allows insertion and removal of elements from both the ends, i.e , front and back.

