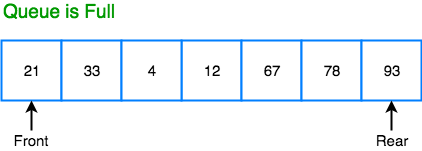
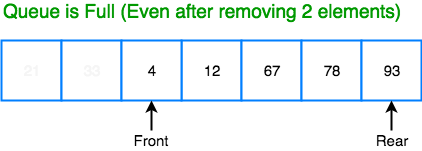
**What is a Circular Queue?**

Before we start to learn about Circular queue, we should first understand, why we need a circular queue, when we already have [linear queue data structure](https://www.studytonight.com/data-structures/queue-data-structure).

In a Linear queue, once the queue is completely full, it's not possible to insert more elements. Even if we dequeue the queue to remove some of the elements, until the queue is reset, no new elements can be inserted. You must be wondering why?



When we **dequeue** any element to remove it from the queue, we are actually moving the **front** of the queue forward, thereby reducing the overall size of the queue. And we cannot insert new elements, because the **rear** pointer is still at the end of the queue.

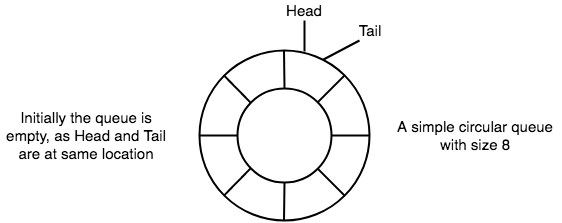


The only way is to reset the linear queue, for a fresh start.

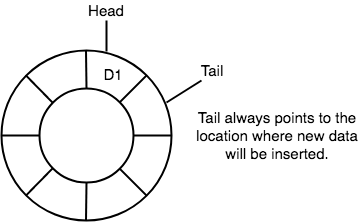
**Circular Queue** is also a linear data structure, which follows the principle of **FIFO**(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

**Basic features of Circular Queue**

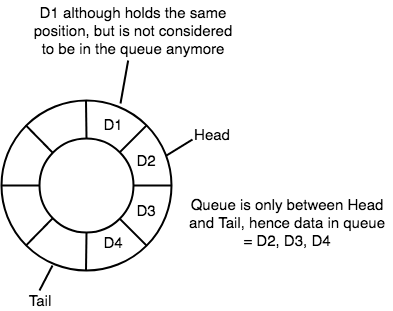
1. In case of a circular queue, head pointer will always point to the front of the queue, and tail pointer will always point to the end of the queue.
2. Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty.



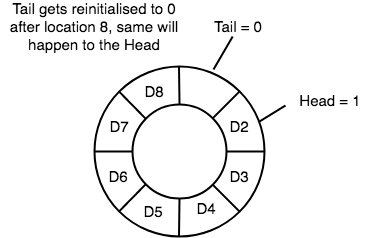
1. New data is always added to the location pointed by the tail pointer, and once the data is added, tail pointer is incremented to point to the next available location.



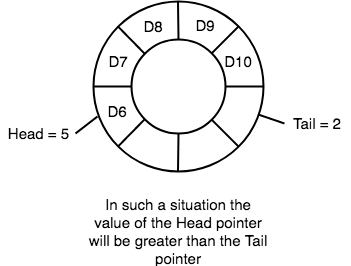
1. In a circular queue, data is not actually removed from the queue. Only the head pointer is incremented by one position when **dequeue** is executed. As the queue data is only the data between head and tail, hence the data left outside is not a part of the queue anymore, hence removed.



1. The head and the tail pointer will get reinitialised to **0** every time they reach the end of the queue.



1. Also, the head and the tail pointers can cross each other. In other words, head pointer can be greater than the tail. Sounds odd? This will happen when we dequeue the queue a couple of times and the tail pointer gets reinitialised upon reaching the end of the queue.



**Going Round and Round**

Another very important point is keeping the value of the tail and the head pointer within the maximum queue size.

In the diagrams above the queue has a size of 8, hence, the value of tail and head pointers will always be between 0 and 7.

This can be controlled either by checking everytime whether tail or head have reached the maxSize and then setting the value 0 or, we have a better way, which is, for a value x if we divide it by 8, the remainder will never be greater than 8, it will always be between 0 and 0, which is exactly what we want.

So the formula to increment the head and tail pointers to make them **go round and round** over and again will be, head = (head+1) % maxSize or tail = (tail+1) % maxSize

**Application of Circular Queue**

Below we have some common real-world examples where circular queues are used:

1. Computer controlled **Traffic Signal System** uses circular queue.
2. CPU scheduling and Memory management.