**QUEUE**

We call the INSERT operation on a queue ENQUEUE, and we call the DELETE operation DEQUEUE; like the stack operation POP, DEQUEUE takes no element argument. The FIFO property of a queue causes it to operate like a line of customers waiting to pay a cashier. The queue has a head and a tail. When an element is enqueued, it takes its place at the tail of the queue, just as a newly arriving customer takes a place at the end of the line. The element dequeued is always the one at the head of the queue, like the customer at the head of the line who has waited the longest.

Queue is an abstract data structure, somewhat 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 queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

**Queue Representation**

As we now understand that in queue, we access both ends for different reasons. The following diagram given below tries to explain queue representation as data structure −

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

**Basic Operations**

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues −

* 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.

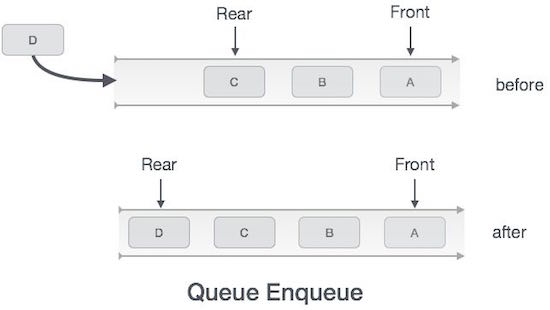
In queue, we always dequeue (or access) data, pointed by front pointer and while enqueing (or storing) data in the queue we take help of rear pointer.

**Enqueue Operation**

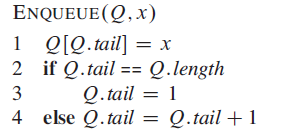
Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue −

* Step 1 − Check if the queue is full.
* Step 2 − If the queue is full, produce overflow error and exit.
* Step 3 − If the queue is not full, increment rear pointer to point the next empty space.
* Step 4 − Add data element to the queue location, where the rear is pointing.
* Step 5 − return success.



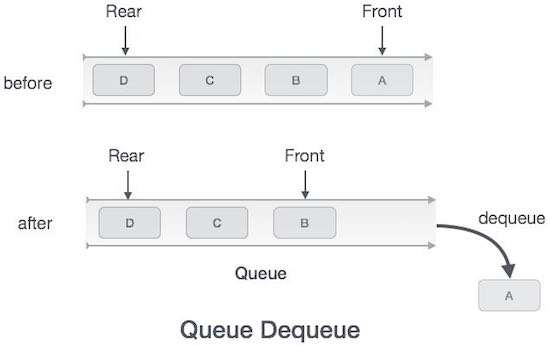
Sometimes, we also check to see if a queue is initialized or not, to handle any unforeseen situations.

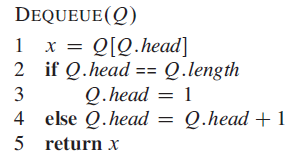


**Dequeue Operation**

Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation −

* Step 1 − Check if the queue is empty.
* Step 2 − If the queue is empty, produce underflow error and exit.
* Step 3 − If the queue is not empty, access the data where front is pointing.
* Step 4 − Increment front pointer to point to the next available data element.
* Step 5 − Return success.





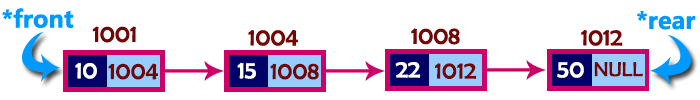
**Queue Using Linked List**

The major problem with the queue implemented using an array is, It will work for an only fixed number of data values. That means, the amount of data must be specified at the beginning itself. Queue using an array is not suitable when we don't know the size of data which we are going to use.

A queue data structure can be implemented using a linked list data structure. The queue which is implemented using a linked list can work for an unlimited number of values. That means, queue using linked list can work for the variable size of data (No need to fix the size at the beginning of the implementation). The Queue implemented using linked list can organize as many data values as we want.

In linked list implementation of a queue, the last inserted node is always pointed by '**rear**' and the first node is always pointed by '**front**'.

**Example**



In above example, the last inserted node is 50 and it is pointed by '**rear**' and the first inserted node is 10 and it is pointed by '**front**'. The order of elements inserted is 10, 15, 22 and 50.

**Operations**

To implement queue using linked list, we need to set the following things before implementing actual operations.

* **Step 1 -**Include all the **header files** which are used in the program. And declare all the **user defined functions**.
* **Step 2 -**Define a '**Node**' structure with two members **data** and **next**.
* **Step 3 -**Define two **Node** pointers '**front**' and '**rear**' and set both to **NULL**.
* **Step 4 -**Implement the **main** method by displaying Menu of list of operations and make suitable function calls in the **main** method to perform user selected operation.

**enQueue(value) - Inserting an element into the Queue**

We can use the following steps to insert a new node into the queue...

* **Step 1 -**Create a **newNode** with given value and set '**newNode → next**' to **NULL**.
* **Step 2 -**Check whether queue is **Empty** (**rear** == **NULL**)
* **Step 3 -**If it is **Empty** then, set **front** = **newNode** and **rear** = **newNode**.
* **Step 4 -**If it is **Not Empty** then, set **rear → next** = **newNode** and **rear** = **newNode**.

**deQueue() - Deleting an Element from Queue**

We can use the following steps to delete a node from the queue...

* **Step 1 -**Check whether **queue** is **Empty** (**front == NULL**).
* **Step 2 -**If it is **Empty**, then display **"Queue is Empty!!! Deletion is not possible!!!"** and terminate from the function
* **Step 3 -**If it is **Not Empty** then, define a Node pointer '**temp**' and set it to '**front**'.
* **Step 4 -**Then set '**front** = **front → next**' and delete '**temp**' (**free(temp)**).

**display() - Displaying the elements of Queue**

We can use the following steps to display the elements (nodes) of a queue...

* **Step 1 -**Check whether queue is **Empty** (**front** == **NULL**).
* **Step 2 -**If it is **Empty** then, display **'Queue is Empty!!!'** and terminate the function.
* **Step 3 -**If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **front**.
* **Step 4 -**Display '**temp → data** --->' and move it to the next node. Repeat the same until '**temp**' reaches to '**rear**' (**temp → next** != **NULL**).
* **Step 5 -**Finally! Display '**temp → data** ---> **NULL**'.

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm>
2. <http://www.btechsmartclass.com/data_structures/queue-using-linked-list.html>

**Lecture Video:**

1. <https://www.youtube.com/watch?v=dPzG0O2ZrPA>
2. <https://www.youtube.com/watch?v=Q4KUd1gmFg0>

**Online Notes:**

1. <http://www.crectirupati.com/sites/default/files/lecture_notes/ds%20ln.pdf>
2. <http://www.vssut.ac.in/lecture_notes/lecture1428550942.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.
2. Lipschutz, S., “*Data Structures, Schaum's Outline Series*”, Tata McGraw Hill.

**Online Book Reference:**

1. <https://www.edutechlearners.com/download/books/DS.pdf>

**In addition: PPT can be also be given.**