- **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
- **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

Now, let's see the ways to implement the queue.
Ways to implement the queue
There are two ways of implementing the Queue:

- **Implementation using array:** The sequential allocation in a Queue can be implemented using an array.
- **Implementation using Linked list:** The linked list allocation in a Queue can be implemented using a linked list.

Operations on each types of Queues: Algorithms and their analysis

Basic Operations for Queue in Data Structure

Unlike arrays and linked lists, elements in the queue cannot be operated from their respective locations. They can only be operated at two data pointers, front and rear. Also, these operations involve standard procedures like initializing or defining data structure, utilizing it, and then wholly erasing it from memory. Here, you must try to comprehend the operations associated with queues:

- **Enqueue()** - Insertion of elements to the queue.
- **Dequeue()** - Removal of elements from the queue.

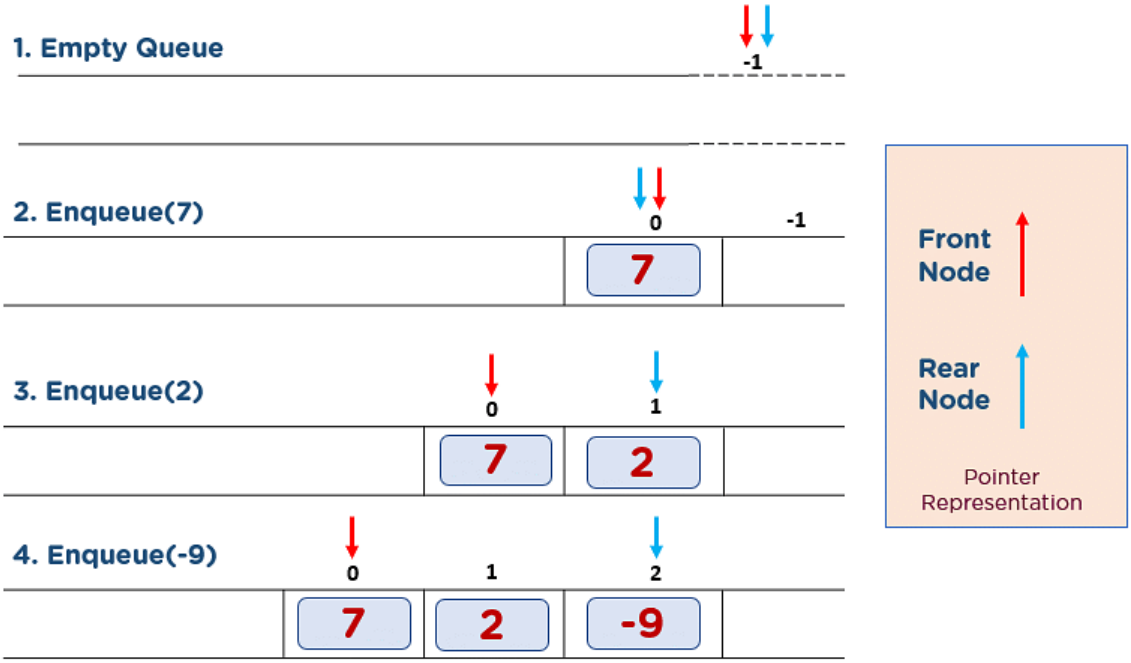
- **Peek()** - Acquires the data element available at the front node of the queue without deleting it.
- **isFull()** - Validates if the queue is full.
- **isNull()** - Checks if the queue is empty.

When you define the queue data structure, it remains empty as no element is inserted into it. So, both the front and rear pointer should be set to -1 (Null memory space). This phase is known as data structure declaration in the context of programming.
First, understand the operations that allow the queue to manipulate data elements in a hierarchy.

Enqueue() Operation

The following steps should be followed to insert (enqueue) data element into a queue -

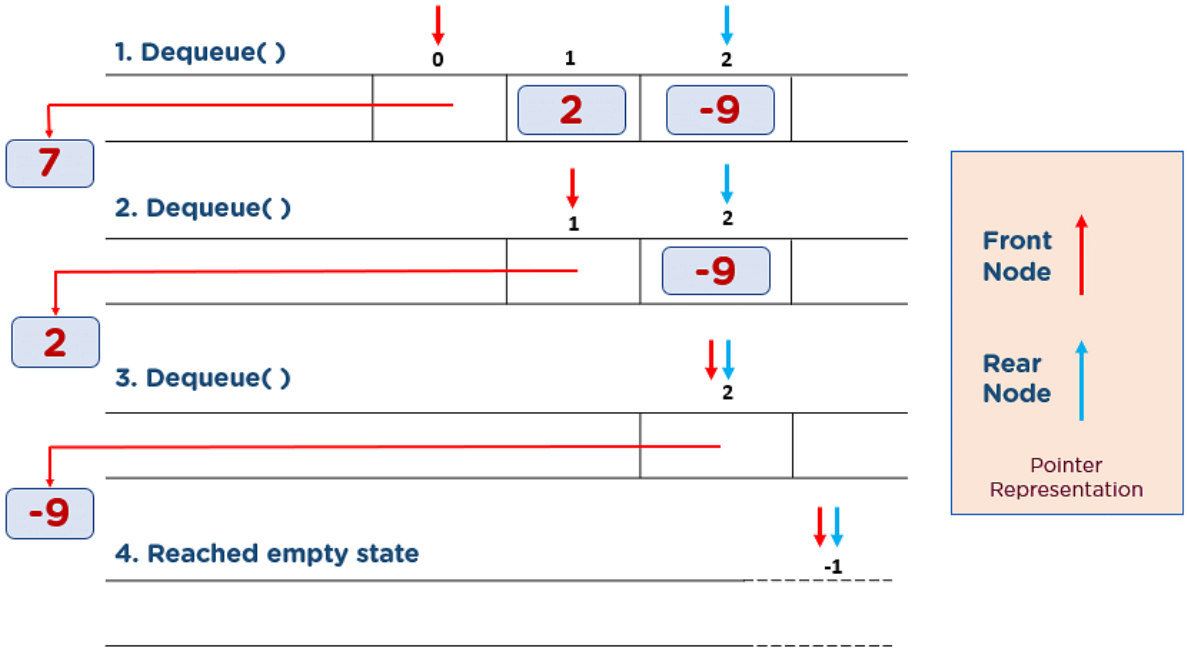
- Step 1: Check if the queue is full.
- Step 2: If the queue is full, Overflow error.
- Step 3: If the queue is not full, increment the rear pointer to point to the next available empty space.
- Step 4: Add the data element to the queue location where the rear is pointing.
- Step 5: Here, you have successfully added 7, 2, and -9.



Dequeue() Operation

Obtaining data from the queue comprises two subtasks: access the data where the front is pointing and remove the data after access. You should take the following steps to remove data from the queue -

- Step 1: Check if the queue is empty.
- Step 2: If the queue is empty, Underflow error.
- Step 3: If the queue is not empty, access the data where the front pointer is pointing.
- Step 4: Increment front pointer to point to the next available data element.
- Step 5: Here, you have removed 7, 2, and -9 from the queue data structure.



Now that you have dealt with the operations that allow manipulation of data entities, you will encounter supportive functions of the queues -

Peek() Operation

This function helps in extracting the data element where the front is pointing without removing it from the queue. The algorithm of Peek() function is as follows-

- Step 1: Check if the queue is empty.
- Step 2: If the queue is empty, return “Queue is Empty.”
- Step 3: If the queue is not empty, access the data where the front pointer is pointing.
- Step 4: Return data.

isFull() Operation

This function checks if the rear pointer is reached at MAXSIZE to determine that the queue is full. The following steps are performed in the isFull() operation -

- Step 1: Check if rear == MAXSIZE - 1.
- Step 2: If they are equal, return “Queue is Full.”

- Step 3: If they are not equal, return “Queue is not Full.”

isNull() Operation

The algorithm of the isNull() operation is as follows -

- Step 1: Check if the rear and front are pointing to null memory space, i.e., -1.
- Step 2: If they are pointing to -1, return “Queue is empty.”
- Step 3: If they are not equal, return “Queue is not empty.”

Algorithms and their analysis in Queue:

Implementation of Queue using LinkedList

```
#include <stdio.h>
#include <stdlib.h>

struct queue{
    int num;
    struct queue *next;
};

struct queue *front= NULL;
struct queue *rear= NULL;

int main(){

    enqueue(10);
    enqueue(20);
    printf("Dequeue : %d\n", dequeue());
    printf("Dequeue : %d\n", dequeue());
```