What is Queue Data Structure?

A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.

Real life examples:

- Waiting in line
- Waiting on hold for tech support

Applications related to Computer

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.

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

FIFO Principle of Queue:

- A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).
- Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the **front** of the queue(sometimes, **head** of the queue), similarly, the position of the last entry in the queue, that is, the

one most recently added, is called the **rear** (or the **tail**) of the queue. See the below figure.

Characteristics of Queue:

- Queue can handle multiple data.
- We can access both ends.
- They are fast and flexible.

Types of Queues:

- **Simple Queue:** Simple queue also known as a linear queue is the most basic version of a queue. Here, insertion of an element i.e. the Enqueue operation takes place at the rear end and removal of an element i.e. the Dequeue operation takes place at the front end. Here problem is that if we pop some item from front and then rear reach to the capacity of the queue and although there are empty spaces before front means the queue is not full but as per condition in isFull() function, it will show that the queue is full then.

- **Circular Queue:** In a circular queue, the element of the queue act as a circular ring. The working of a circular queue is similar to the linear queue except for the fact that the last element is connected to the first element. Its advantage is that the memory is utilized in a better way. This is because if there is an empty space i.e. if no element is present at a certain position in the queue, then an element can be easily added at that position using modulo capacity(*%n*).

- **Priority Queue:** This queue is a special type of queue. Its specialty is that it arranges the elements in a queue based on some priority. The priority can be something where the element with the highest value has the priority so it creates a queue with decreasing order of values. The priority can also be such that the element with the lowest value gets the highest priority so in turn it creates a queue with increasing order of values.

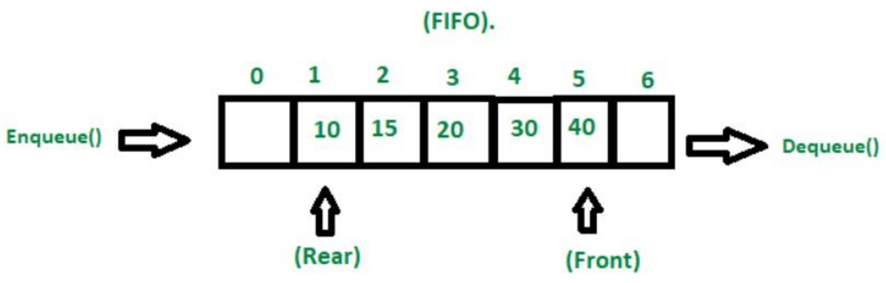
- **Dequeue:** Dequeue is also known as Double Ended Queue. As the name suggests double ended, it means that an element can be inserted or removed from both ends of the queue, unlike the other queues in which it can be done only from one end. Because of this property, it may not obey the First In First Out property.

Representation of Queue

1. Array Representation of Queue:

Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are

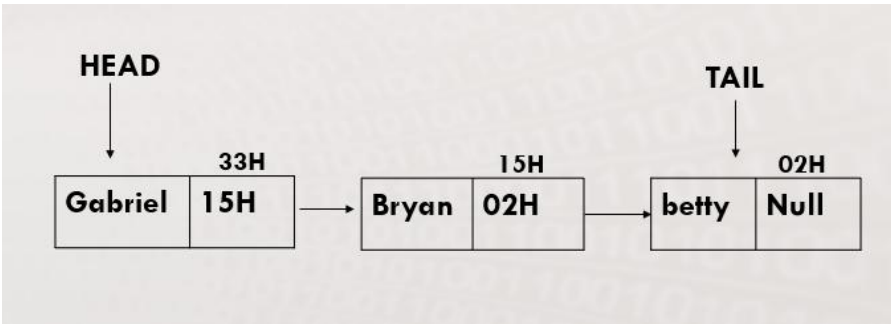
- **Queue:** the name of the array storing queue elements.
- **Front**: the index where the first element is stored in the array representing the queue.
- **Rear:** the index where the last element is stored in an array representing the queue.



2. Queue as Singly-linked List

When using singly-linked list to represent a queue, all insertion operations are performed at the TAIL, while all deletion and retrieval operations are done from the HEAD.

Queues are represented as singly-linked lists when the maximum size is known.



Basic Operations on Queue

Like arrays and stacks, we also have the INSERT, REMOVE and PEEK operations on queue, we only use different terms.

ARRAYS	STACKS	QUEUES
Insert	**Push**	**Put, Add. Enqueue**
Delete	**Pop**	**Delete, Get, Dequeue**
Search	**Peek**	**Peek, Retrieve**

- **Enqueue**

  Inserts an element at the end of the queue i.e. at the rear end. 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.

- **Dequeue**

  This operation removes and returns an element that is at the front end of the queue. 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.

- **Peek or Retrieve**

  The retrieving of element in the queue is to simply print or display the element at the bottom of the list. Basically, what is being viewed or retrieved is what you are about to delete remove and not what you have inserted.
- **Empty Or Full**

  If in any case that the list is EMPTY and you are trying to retrieve element from the list, it should then return an error because the queue is empty. On the other hand, it can also give you a message that the queue is FULL when trying to insert another element.

Implementation of Queues in C#

Queue represents a ***first-in, first out*** collection of object. It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called **enqueue**, and when you remove an item, it is called **dequeue** . This class comes under **System.Collections** namespace and implements *ICollection, IEnumerable, and ICloneable* interfaces.

Characteristics of Queue Class:

- **Enqueue** adds an element to the end of the Queue.
- **Dequeue** removes the oldest element from the start of the Queue.
- **Peek** returns the oldest element that is at the start of the Queue but does not remove it from the Queue.
- The **capacity** of a Queue is the number of elements the Queue can hold.
- As elements are added to a Queue, the capacity is automatically increased as required by reallocating the internal array.
- Queue accepts **null** as a valid value for reference types and allows duplicate elements.

Queue Constructors

CONSTRUCTOR	DESCRIPTION
Queue()	Initializes a new instance of the Queue class that is empty, has the default initial capacity, and uses the default growth factor.
Queue(ICollection)	Initializes a new instance of the Queue class that contains elements copied from the specified collection, has the same initial capacity as the number of elements copied, and uses the default growth factor.
Queue(Int32)	Initializes a new instance of the Queue class that is empty, has the specified initial capacity, and uses the default growth factor.
Queue(Int32, Single)	Initializes a new instance of the Queue class that is empty, has the specified initial capacity, and uses the specified growth factor.

How to create an Queue using Queue() constructor

Step 1: Include *System.Collections* namespace in your program with the help of using keyword.

Syntax:

using System.Collections;

Step 2: Create an queue using Queue class as shown below:

Queue queue_name = new Queue();

Step 3: If you want to add elements in your queue then use ***Enqueue()*** method to add elements in your queue. As shown in the below example.

How to remove elements from the Queue?

In Queue, you are allowed to remove elements from the queue. The Queue class provides two different methods to remove elements and the methods are:

- **Clear**: This method is used to remove the objects from the queue.
- **Dequeue**: This method removes the beginning element of the queue.

How to get topmost element of the queue?

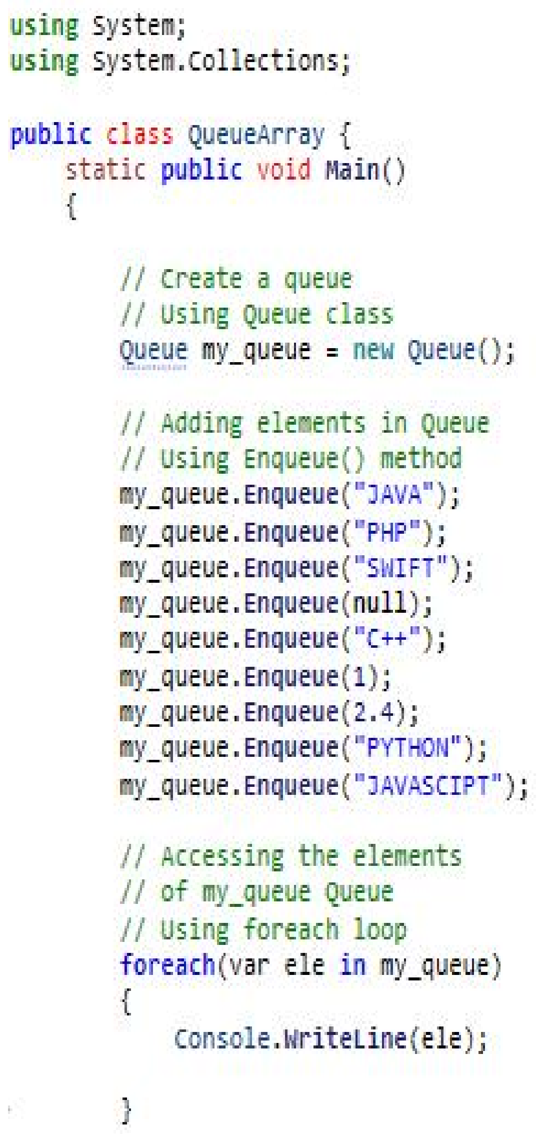
In Queue, you can easily find the topmost element of the queue by using the following methods provided by the Queue class:

- **Peek**: This method returns the object at the beginning of the Queue without removing it.
- **Dequeue:** This method returns the object at the beginning of the Queue with modification means this method remove the topmost element of the queue.

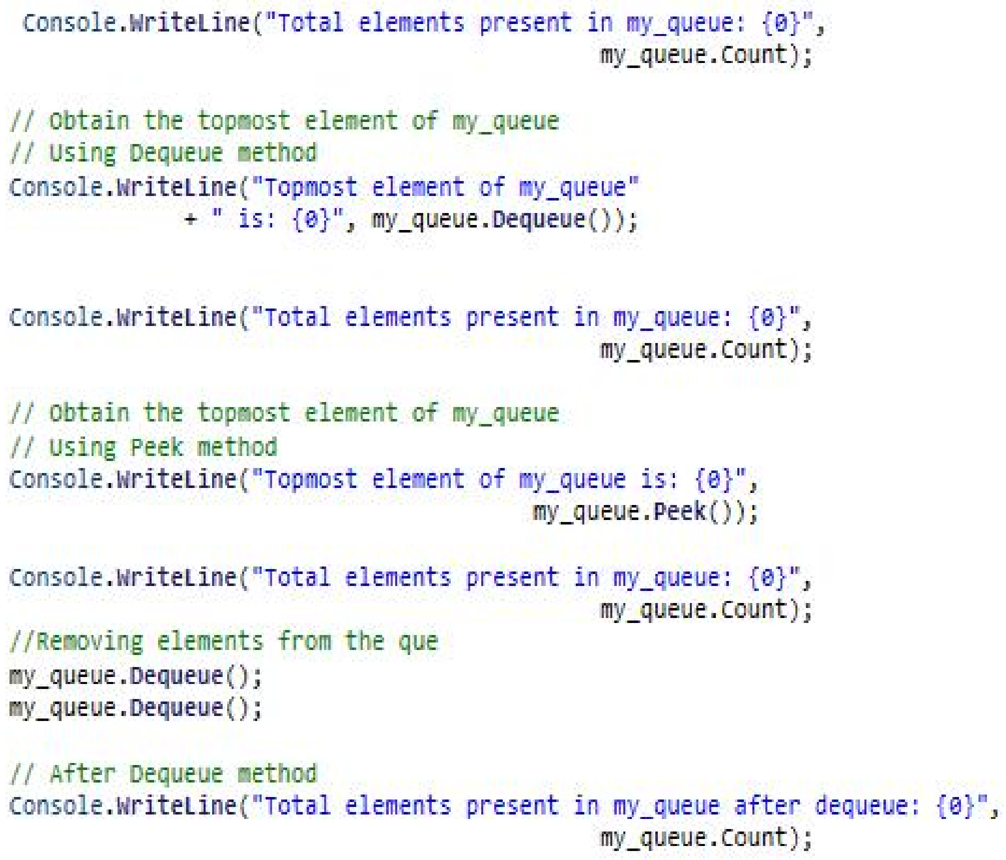
How to check the availability of elements in the queue?

In Queue, you can check whether the given element is present or not using Contain() method. Or in other words, if you want to search an element in the given queue use ***Contains()*** method.

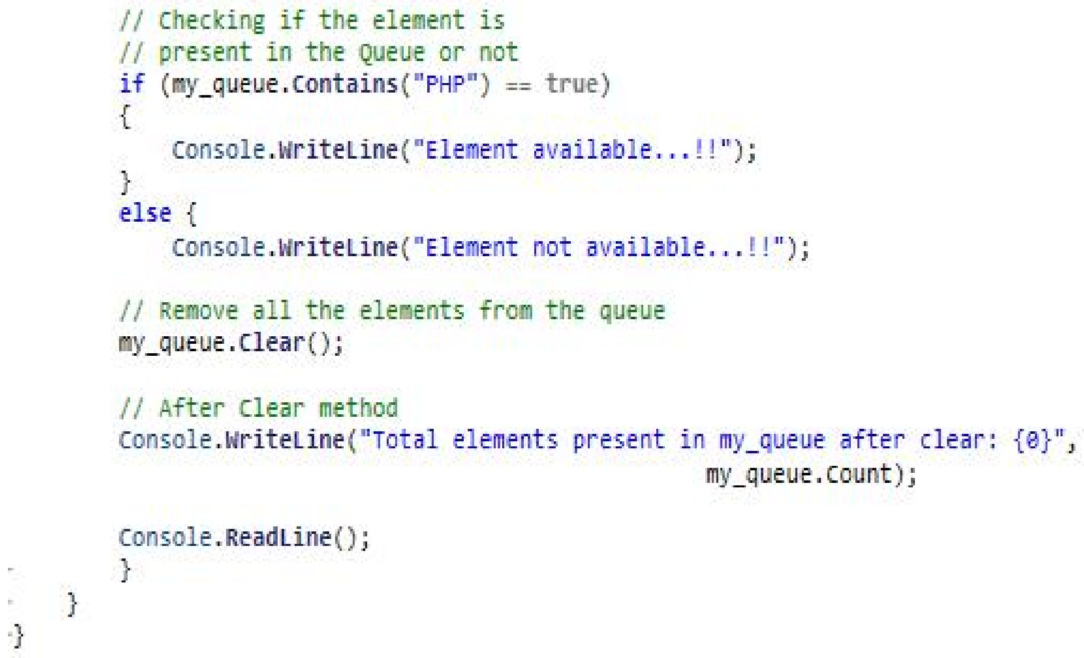
Example Program



```
using System;
using System.Collections;

public class QueueArray {
    static public void Main()
    {

        // Create a queue
        // Using Queue class
        Queue my_queue = new Queue();

        // Adding elements in Queue
        // Using Enqueue() method
        my_queue.Enqueue("JAVA");
        my_queue.Enqueue("PHP");
        my_queue.Enqueue("SWIFT");
        my_queue.Enqueue(null);
        my_queue.Enqueue("C++");
        my_queue.Enqueue(1);
        my_queue.Enqueue(2.4);
        my_queue.Enqueue("PYTHON");
        my_queue.Enqueue("JAVASCIPT");

        // Accessing the elements
        // of my_queue Queue
        // Using foreach loop
        foreach(var ele in my_queue)
        {
            Console.WriteLine(ele);

        }
```

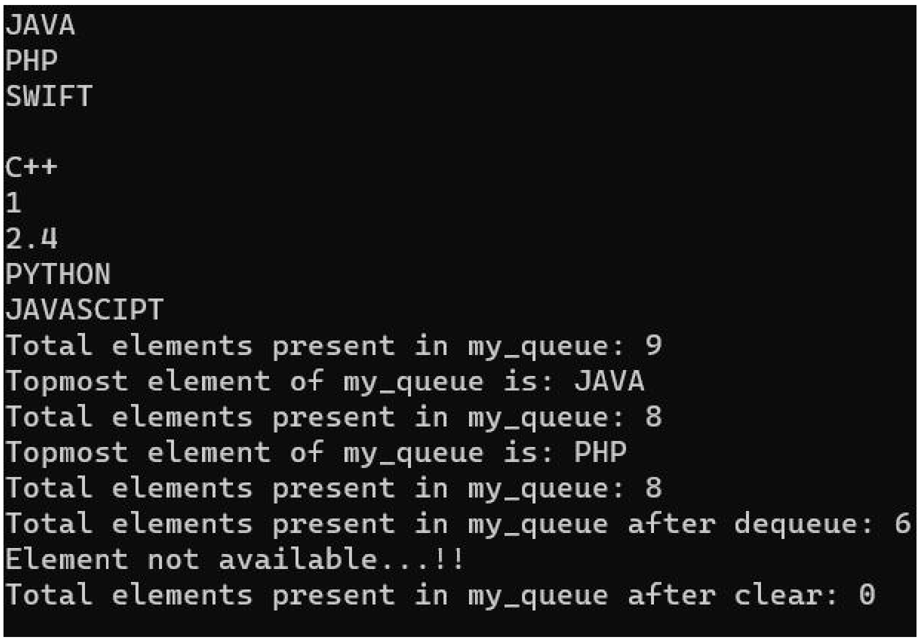


```
             Console.WriteLine("Total elements present in my_queue: {0}",
                                                  my_queue.Count);

            // Obtain the topmost element of my_queue
            // Using Dequeue method
            Console.WriteLine("Topmost element of my_queue"
                          + " is: {0}", my_queue.Dequeue());


            Console.WriteLine("Total elements present in my_queue: {0}",
                                                  my_queue.Count);

            // Obtain the topmost element of my_queue
            // Using Peek method
            Console.WriteLine("Topmost element of my_queue is: {0}",
                                            my_queue.Peek());

            Console.WriteLine("Total elements present in my_queue: {0}",
                                                  my_queue.Count);
            //Removing elements from the que
            my_queue.Dequeue();
            my_queue.Dequeue();

            // After Dequeue method
            Console.WriteLine("Total elements present in my_queue after dequeue: {0}",
                                                  my_queue.Count);
```



```
             // Checking if the element is
             // present in the Queue or not
             if (my_queue.Contains("PHP") == true)
             {
                 Console.WriteLine("Element available...!!");
             }
             else {
                 Console.WriteLine("Element not available...!!");

             // Remove all the elements from the queue
             my_queue.Clear();

             // After Clear method
             Console.WriteLine("Total elements present in my_queue after clear: {0}",
                                                   my_queue.Count);

             Console.ReadLine();
             }
     }
}
```

OUTPUT



```
JAVA
PHP
SWIFT

C++
1
2.4
PYTHON
JAVASCIPT
Total elements present in my_queue: 9
Topmost element of my_queue is: JAVA
Total elements present in my_queue: 8
Topmost element of my_queue is: PHP
Total elements present in my_queue: 8
Total elements present in my_queue after dequeue: 6
Element not available...!!
Total elements present in my_queue after clear: 0
```