Queue | Set 1 (Introduction and Array Implementation)

Like Stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Operations on Queue:

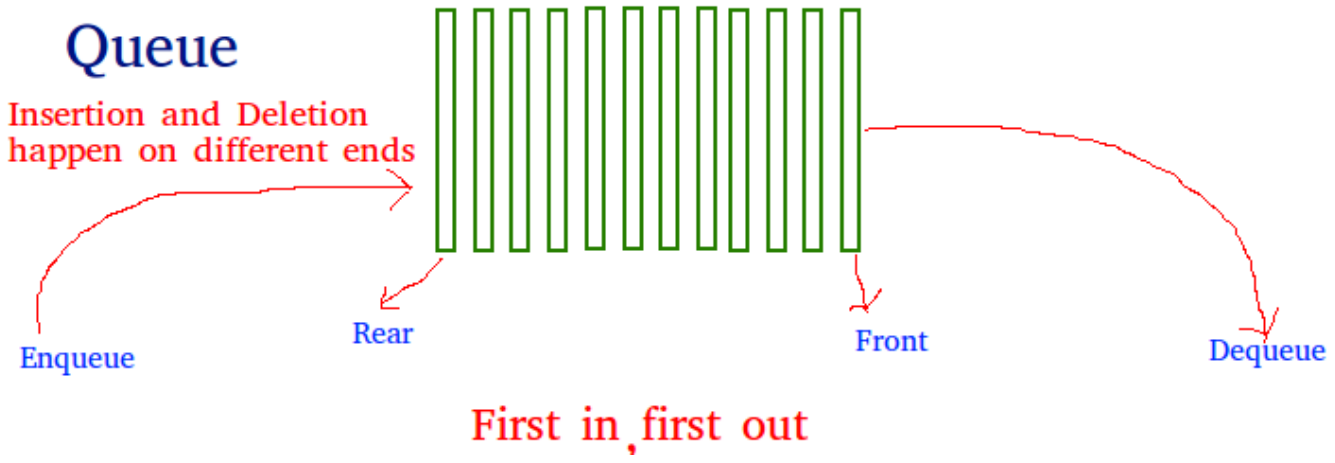
Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the front item from queue.

Rear: Get the last item from queue.



Applications of Queue:

Queue is used when things don't have to be processed immediatly, but have to be processed in **F**irst **I**n **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

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

2) 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.

See this for more detailed applications of Queue and Stack.

Array implementation Of Queue

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front. If we simply increment front and rear indices, then there may be problems, front may reach end of the array. The solution to this problem is to increase front and rear in circular manner (See this for details)

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

C

```c
// C program for array implementation of queue
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// A structure to represent a queue
struct Queue
{
    int front, rear, size;
    unsigned capacity;
    int* array;
};

// function to create a queue of given capacity.
// It initializes size of queue as 0
struct Queue* createQueue(unsigned capacity)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;  // This is important, see the enqueue
    queue->array = (int*) malloc(queue->capacity * sizeof(int));
    return queue;
}

// Queue is full when size becomes equal to the capacity
int isFull(struct Queue* queue)
{  return (queue->size == queue->capacity);  }

// Queue is empty when size is 0
int isEmpty(struct Queue* queue)
{  return (queue->size == 0); }

// Function to add an item to the queue.
// It changes rear and size
void enqueue(struct Queue* queue, int item)
{
    if (isFull(queue))
        return;
```

```
    queue->rear = (queue->rear + 1)%queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    printf("%d enqueued to queue\n", item);
}

// Function to remove an item from queue.
// It changes front and size
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1)%queue->capacity;
    queue->size = queue->size - 1;
    return item;
}

// Function to get front of queue
int front(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->front];
}

// Function to get rear of queue
int rear(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->rear];
}

// Driver program to test above functions./
int main()
{
    struct Queue* queue = createQueue(1000);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("%d dequeued from queue\n\n", dequeue(queue));

    printf("Front item is %d\n", front(queue));
    printf("Rear item is %d\n", rear(queue));

    return 0;
}
```

Run on IDE | Copy Code

Java

```
// Java program for array implementation of queue

// A class to represent a queue
class Queue
{
    int front, rear, size;
    int  capacity;
    int array[];

    public Queue(int capacity) {
         this.capacity = capacity;
         front = this.size = 0;
         rear = capacity - 1;
         array = new int[this.capacity];
```

```
    }

    // Queue is full when size becomes equal to
    // the capacity
    boolean isFull(Queue queue)
    {  return (queue.size == queue.capacity);
    }

    // Queue is empty when size is 0
    boolean isEmpty(Queue queue)
    {  return (queue.size == 0); }

    // Method to add an item to the queue.
    // It changes rear and size
    void enqueue( int item)
    {
        if (isFull(this))
            return;
        this.rear = (this.rear + 1)%this.capacity;
        this.array[this.rear] = item;
        this.size = this.size + 1;
        System.out.println(item+ " enqueued to queue");
    }

    // Method to remove an item from queue.
    // It changes front and size
    int dequeue()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        int item = this.array[this.front];
        this.front = (this.front + 1)%this.capacity;
        this.size = this.size - 1;
        return item;
    }

    // Method to get front of queue
    int front()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        return this.array[this.front];
    }

    // Method to get rear of queue
    int rear()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        return this.array[this.rear];
    }
}


// Driver class
public class Test
{
    public static void main(String[] args)
    {
        Queue queue = new Queue(1000);

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);

        System.out.println(queue.dequeue() +
                    " dequeued from queue\n");

        System.out.println("Front item is " +
                                queue.front());
```

```
        System.out.println("Rear item is " +
                                queue.rear());
    }
}

// This code is contributed by Gaurav Miglani
```

Run on IDE | Copy Code

Python3

```
# Python3 program for array implementation of queue

# Class Queue to represent a queue
class Queue:

    # __init__ function
    def __init__(self, capacity):
        self.front = self.size = 0
        self.rear = capacity -1
        self.Q = [None]*capacity
        self.capacity = capacity

    # Queue is full when size becomes
    # equal to the capacity
    def isFull(self):
        return self.size == self.capacity

    # Queue is empty when size is 0
    def isEmpty(self):
        return self.size == 0

    # Function to add an item to the queue.
    # It changes rear and size
    def EnQueue(self, item):
        if self.isFull():
            print("Full")
            return
        self.rear = (self.rear + 1) % (self.capacity)
        self.Q[self.rear] = item
        self.size = self.size + 1
        print("%s enqueued to queue"  %str(item))

    # Function to remove an item from queue.
    # It changes front and size
    def DeQueue(self):
        if self.isEmpty():
            print("Empty")
            return

        print("%s dequeued from queue" %str(self.Q[self.front]))
        self.front = (self.front + 1) % (self.capacity)
        self.size = self.size -1

    # Function to get front of queue
    def que_front(self):
        if self.isEmpty():
            print("Queue is empty")

        print("Front item is", self.Q[self.front])

    # Function to get rear of queue
    def que_rear(self):
        if self.isEmpty():
            print("Queue is empty")
        print("Rear item is",  self.Q[self.rear])


# Driver Code
if __name__ == '__main__':
```

```
queue = Queue(30)
queue.EnQueue(10)
queue.EnQueue(20)
queue.EnQueue(30)
queue.EnQueue(40)
queue.DeQueue()
queue.que_front()
queue.que_rear()
```

Run on IDE Copy Code

C#

```
// C# program for array implementation of queue
using System;

namespace GeeksForGeeks
{
    // A class to represent a linearqueue
    class Queue
    {
        private int []ele;
        private int front;
        private int rear;
        private int max;

        public Queue(int size)
        {
            ele = new int[size];
            front = 0 ;
            rear = -1;
            max = size;
        }

        // Function to add an item to the queue.
        // It changes rear and size
        public void enqueue(int item)
        {
            if (rear == max-1)
            {
                Console.WriteLine("Queue Overflow");
                return;
            }
            else
            {
                ele[++rear] = item;
            }

        }

        // Function to remove an item from queue.
        // It changes front and size
        public int dequeue()
        {
            if(front == rear + 1)
            {
                Console.WriteLine("Queue is Empty");
                return -1;
            }
            else
            {
                Console.WriteLine( ele[front]+" dequeued from queue");
                int p = ele[front++];
                Console.WriteLine();
                Console.WriteLine("Front item is {0}",ele[front]);
                Console.WriteLine("Rear item is {0} ",ele[rear]);
        return p;
            }

        }
```

```
        // Function to print queue.
        public void printQueue()
        {
            if (front == rear + 1)
            {
                Console.WriteLine("Queue is Empty");
                return;
            }
            else
            {
                for (int i = front; i <= rear; i++)
                {
                    Console.WriteLine(ele[i]+ " enqueued to queue" );
                }
            }

        }
    }

    // Driver code
    class Program
    {
        static void Main()
        {
            Queue Q = new Queue(5);

            Q.enqueue(10);
            Q.enqueue(20);
            Q.enqueue(30);
            Q.enqueue(40);
            Q.printQueue();
            Q.dequeue();


        }
    }
}
```

Run on IDE | Copy Code

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```

Time Complexity: Time complexity of all operations like enqueue(), dequeue(), isFull(), isEmpty(), front() and rear() is O(1). There is no loop in any of the operations.