# **Tutorial 6 (Data Structure)**

**Name**: Krishna Pandey

**Admission Number**: U20CS110

**Roll No**: B110

1)

a) Enqueue Algorithm

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6. Check if the queue is full.

7. If the queue is full, produce an overflow error and exit.

8.  If the queue is not full, increment the rear pointer to point to the next empty space.

9.Add the data element to the queue location, where the rear is pointing.

10.return success.

11.Stop

b) Dequeue Algorithm

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If the queue is empty, produce an underflow error and exit.

8. If the queue is not empty, access the data where the frontis pointing.

**9.** Increment frontpointer to point to the next available data element.

10**.** Return success.

11.Stop

c)Peek Algorithm

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If the queue is empty, produce an underflow error and exit.

8. Begin procedure peek

9.return queue[front]

10.end procedure.

11.Stop

d) isfull Algorithm

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is full.

**7.** If the queue is empty, produce an underflow error and exit.

8. By using if loop with condition rear == MAX-1

9.return the result.

10.end procedure.

11.Stop

e) isempty Algorithm

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If the queue is empty, produce an underflow error and exit.

8. By using if loop with condition front == -1.

9.return the result.

10.end procedure.

11.Stop.

2. Write Algorithm for Dequeue (double ended queue)

a) Insert element at back

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If the queue is empty then intialize front and rear to 0. Both will point to the first element..

8. Else we increment rear and insert the element. Since we are using circular array, we have to keep in mind that if rear is equal to SIZE-1 then instead of increasing it by 1 we make it equal to 0.

9.return the result.

10.Stop.

b) Insert element at front

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If the queue is empty then intialize front and rear to 0. Both will point to the first element.

8 Else we decrement front and insert the element. Since we are using circular array, we have to keep in mind that if front is equal to 0 then instead of decreasing it by 1 we make it equal to SIZE-1.

9.return the result.

10.Stop.

c) Remove element at front

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If only one element is present we once again make front and rear equal to -1.

8. Else we increment front. But we have to keep in mind that if front is equal to SIZE-1 then instead of increasing it by 1 we make it equal to 0.9.return the result.

10.Stop.

c) Remove element at back

1.Start

2.Intialize an array of given size.

3. Initialize a queue structure with two pointers front and rear.

4. Front and rear will be initialized to -1 .

5. As we insert elements in queue, we increment rear by 1.

6  Check if the queue is empty.

**7.** If only one element is present, we make front and rear equal to -1.

8. Else we decrement rear. But we have to keep in mind that if rear is equal to 0 then instead of decreasing it by 1 we make it equal to SIZE-1..

10.Stop.