INTRODUCTION TO DATA STRUCTURES & ALGORITHMS

OMAR AL-MUKHTAR UNIVERSITY

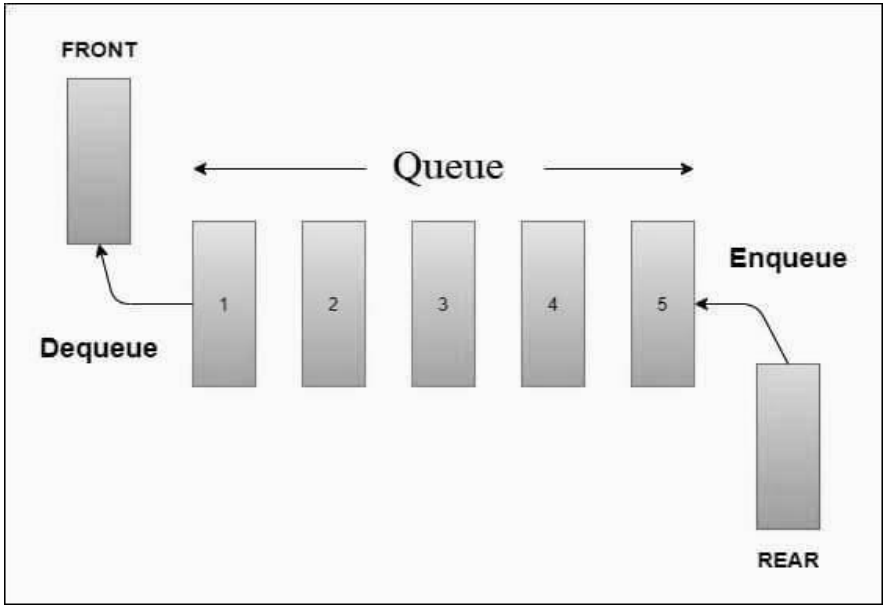
FACULTY OF SCIENCE

COMPUTER SCIENCE DEPARTMENT

LECTURE 3

THE QUEUE

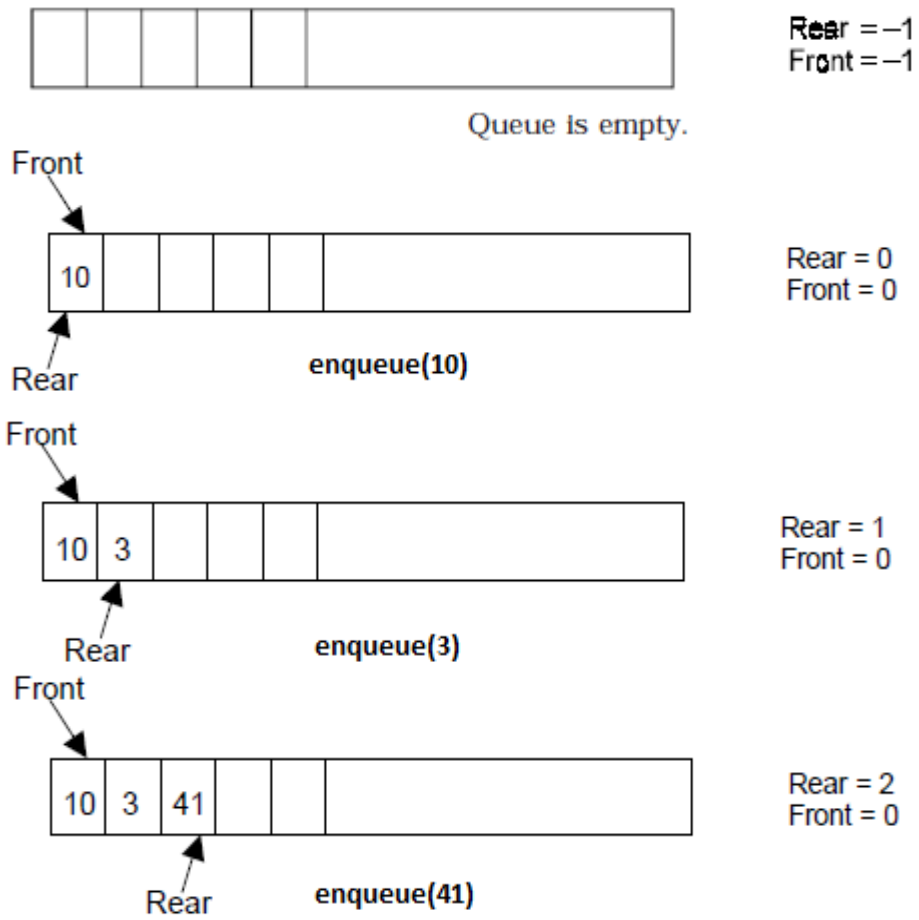
- A Queue is a linear data structure which follows a particular order first in first out (FIFO or first come first serve).
- a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).
- i.e., First-In-First-Out methodology, the data item stored first will be accessed first.



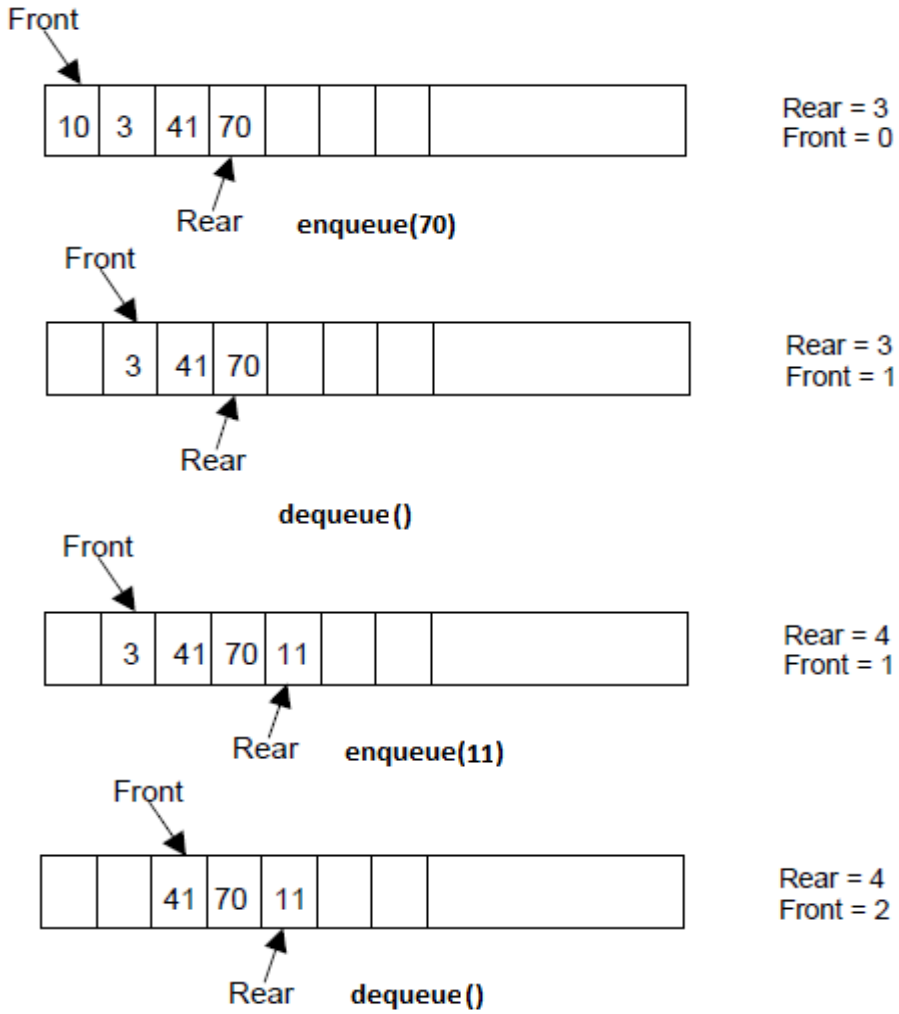
QUEUE ADT

- A real-world example of queue can be a one-way road, where the car enters first, exits first.
- More real-world examples can be seen as queues at the ticket windows and bus-stops.

QUEUE BASIC OPERATIONS



QUEUE BASIC OPERATIONS



BASIC OPERATIONS PERFORMED ON QUEUE

- The basic operations that can be performed on a queue are:

1. Insert ((Enqueue)) − an element to the queue at the rear end, by incrementing the array index.
2. Delete ((Dequeue)) − an element from a queue (dequeue) from the front end by incrementing the array index " decrementing the range of array ".

- *If we try to pop (or delete or remove) an element from queue when it is *empty*, underflow occurs. If we try to push (or insert or add) an element to queue when queue is *full*, overflow occurs.

QUEUE IMPLEMENTATION

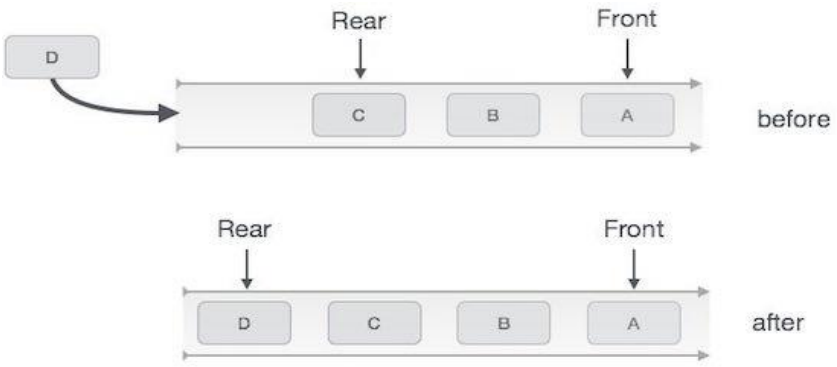
- A Queue can be implemented in two ways:
- 1. Using arrays (static).
- 2. Using pointers (dynamic).

- In implementation of the static Queue, an array will be used so all operation of queue are index based. A Static Queue is a queue of fixed size implemented using array.

QUEUE USING ARRAYS

- Enqueue Operation Steps:

- 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 index to the next empty space.
- Step 4 − Add data element to the queue location, where the rear is pointing.
- Step 5 − Exit.



Queue Enqueue

QUEUE USING ARRAYS

- Algorithm for INSERTING AN ELEMENT :

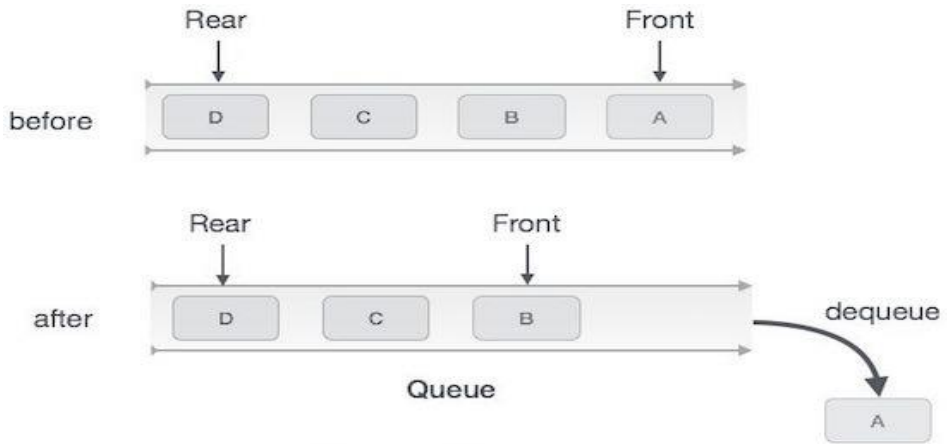
Let Q be the array of some specified size say SIZE.

1. Initialize Front= –1 Rear = –1
2. Input the value to be inserted and assign to variable “data”
3. If (Rear = SIZE –1) , then:
   (a) Display “Queue overflow”
   (b) Exit
4. If Rear = –1 ,then
   (a) Front = 0
   (b) Rear = 0
   Else
   Rear = Rear +1
5. Q[Rear] = data
6. Exit

QUEUE USING ARRAYS

- Dequeue Operation Steps:

- 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 index to point to the next available data element.
- Step 5 − Exit.



Queue Dequeue

QUEUE USING ARRAYS

- Algorithm for DELETING AN ELEMENT:

Let Q be the array of some specified size say SIZE.

1. If (Front = –1)
   (a) Display “The queue is empty”
   (b) Exit
2. data = Q[Front]
3. If Front = Rear , then:
   (a) Front = –1
   (b) Rear = –1
   Else
   Front = front +1
4. Exit

APPLICATIONS OF QUEUE

- Applications related to computers:

1. Queue concept is used in printers. A print queue on a computer has print jobs that are waiting to be sent to the print processor.
2. OS Job scheduling (FIFO Scheduling).
3. Keyboard buffer, the letters appear on the screen in the order you press them.
4. Message processing in client server applications in computer networks.

DISADVANTAGES OF QUEUE

- On deletion of an element from existing queue, front index is shifted to next position. This results into virtual deletion of an element, and by doing so memory space which was occupied by deleted element is wasted and hence inefficient memory utilization is occur.

- To overcome disadvantage of linear queue, circular queue is use.

- We can solve this problem by joining the front and rear end of a queue to make the queue as a circular queue .

APPLICATIONS OF QUEUE

- CIRCULAR QUEUE: is an abstract data structure in which the operations are performed based on FIFO (First In First Out) rule and the last position is connected back just to the first position to making a circle.

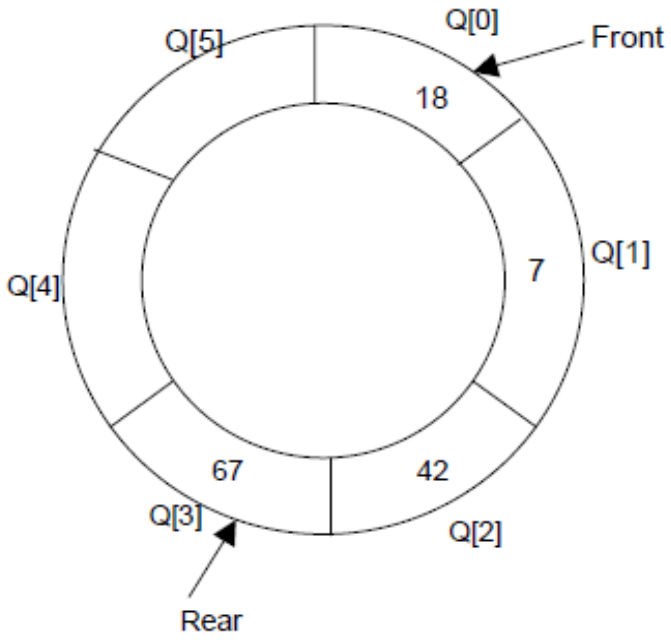
- Why Circular Queue?

- In the ordinary Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue. To overcome this problem, the circular queue data structure is used.

- A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.

CIRCULAR QUEUE

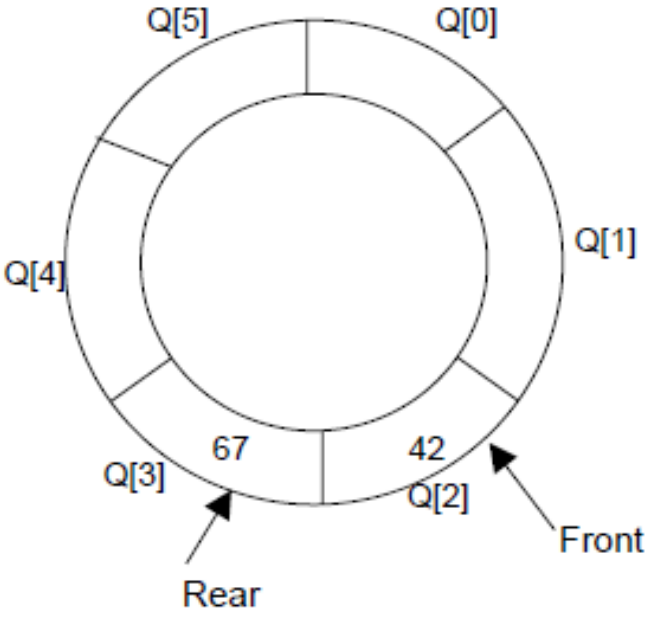
- Suppose Q is a queue array of 6 elements. enqueue and dequeue operations can be performed as shown:



- A circular queue after inserting 18, 7, 42, 67.

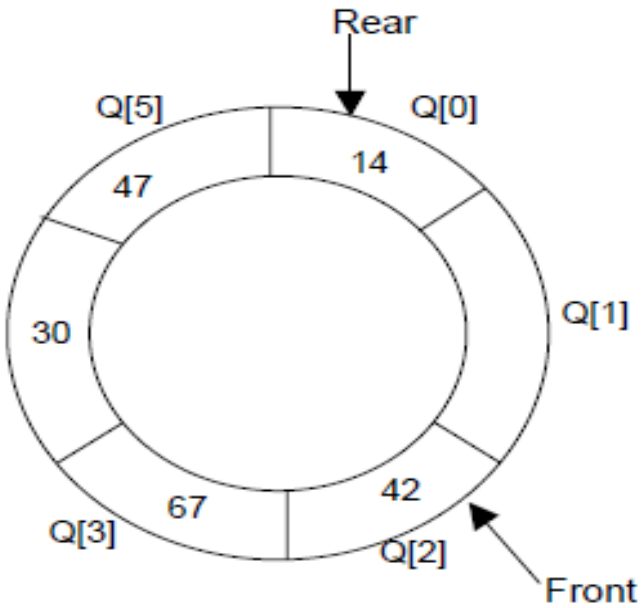
CIRCULAR QUEUE

- A circular queue after dequeue 18, 7



CIRCULAR QUEUE

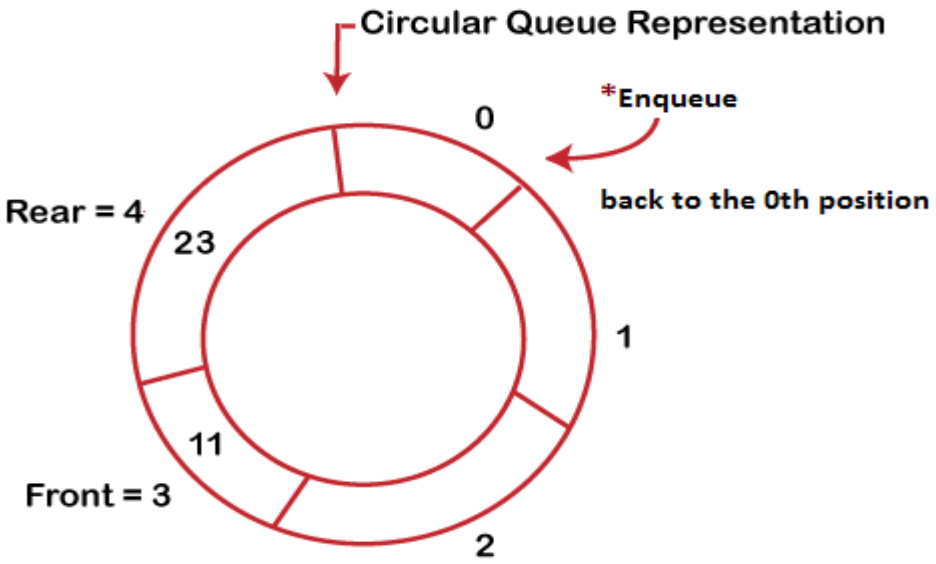
- *After inserting an element at last location Q[5], the next element will be inserted at the very first location (i.e., Q[0])



A circular queue after enqueue 30, 47, 14

CIRCULAR QUEUE

- After rear reaches the last position, i.e. MAX-1 in order to reuse the vacant positions, we can bring rear back to the 0th position, if it is empty, and continue incrementing rear in same way as earlier.
- Thus rear will have to be incremented circularly.



- For deletion, front will also have to be incremented circularly..

CIRCULAR QUEUE

- In order to get to the very first location the modulo operator '**%**' is used.
- At any time the position of the element to be inserted will be calculated by the relation Rear = (Rear + 1) % SIZE
- After deleting an element from circular queue the position of the front end is calculated by the relation Front= (Front + 1) % SIZE
- *This operator '%' keeps you moving circularly, from 0 to n-1 and then ->> to zero again.*

NOTE.."n= max size"

ADVANTAGES OF CIRCULAR QUEUE

- In Circular Queues we utilize memory efficiently. because in ordinary queue when we delete any element only front increment by 1, but that position is not used later. so when we perform more add and delete operation, memory wastage increase. But in Circular Queue memory is utilized, if we delete any element that position is used later, because it is circular.

CIRCULAR QUEUE ALGORITHMS

Suppose **Q** be the array of some specified size say SIZE. **FRONT** and **REAR** are two pointers where the elements are deleted and inserted at two ends of the circular queue. DATA is the element to be inserted.

1. Algorithm of Inserting an element to circular Queue:

1) Initialize FRONT = – 1; REAR = – 1
2) If (FRONT is equal to (REAR + 1) % SIZE) // Queue is full
   - (a) Display "Queue is full"
   - (b) Exit
   - Input the value to be inserted and assign to variable "DATA"
5) If (FRONT is equal to – 1) // Queue is empty
   - (a) FRONT = 0
   - (b) REAR = 0
6) Else

   REAR = (REAR + 1) % SIZE
7) Q[REAR] = DATA
8) Repeat steps 2 to 5 if we want to insert more elements
9) Exit

CIRCULAR QUEUE ALGORITHMS

2. Algorithm of Deleting an element to circular Queue:

1) If (FRONT is equal to – 1) // Queue is empty

   (a) Display "Queue is empty"

   (b) Exit

2) Else

- DATA = Q[FRONT]

3) If (REAR is equal to FRONT) // Only one element in the Queue

- (a) FRONT = –1
- (b) REAR = –1

4) Else

- FRONT = (FRONT +1) % SIZE

5) Repeat the steps 1, 2 and 3 if we want to delete more elements

6) Exit

(LINEAR / CIRCULAR) QUEUE

#	LINEAR QUEUE	CIRCULAR QUEUE
1	A linear data structure that stores data as a sequence of element similar to a real world queue.	A linear data structure in which the last item connects back to the first item forming a circle.
2	Possible to enter new items from the rear end and remove the items from the front.	Possible to enter and remove elements from any position.
3	Requires more memory.	Requires less memory.
4	Less efficient.	More efficient.

STACK VS QUEUE

#	STACK	QUEUE
1	Objects are inserted and removed at the same end.	Objects are inserted and removed from different ends.
2	In stacks only one pointer is used. It points to the top of the stack.	In queues, two different pointers are used for front and rear ends.
3	In stacks, the last inserted object is first to come out.	In queues, the object inserted first is first deleted.
4	Stacks follow Last In First Out (LIFO) order.	Queues following First In First Out (FIFO) order.
5	Stack operations are called push and pop.	Queue operations are called enqueue and dequeue.
6	Stacks are visualized as vertical collections.	Queues are visualized as horizontal collections.
7	Collection of dinner plates is an example of stack.	People standing in a line to board a bus is an example of queue.