Data Structure

Queues

Lecturer: Dr. Alaa Ahmed Abbood
Lecture 8.
Class 2nd .
Time: 12:30-2:30
Department: Businesses Information Technology (BIT)

Outline

- Queues
  - The Queue Abstract Data Type
  - Representation Of Queues In Memory
  - Algorithms for Insert and Delete Operations in Linear Queue
  - Drawback of Linear Queue
  - Circular Queue
  - Algorithms for Insert and Delete Operations in Circular Queue

Queues

- Another fundamental data structure is the queue, which is a close relative of the stack. A queue is a container of elements that are inserted and removed according to the first-in first-out (FIFO) principle. Elements can be inserted in a queue at any time, but only the element that has been in the queue the longest can be removed at any time. We usually say that elements enter the queue at the rear and are removed from the front. The metaphor for this terminology is a line of people waiting to get on an amusement park ride. People enter at the rear of the line and get on the ride from the front of the line as showing in the figure (Next slide).

Queues

Rear of Queue
Front of Queue
next...
PAY HERE

The Queue Abstract Data Type

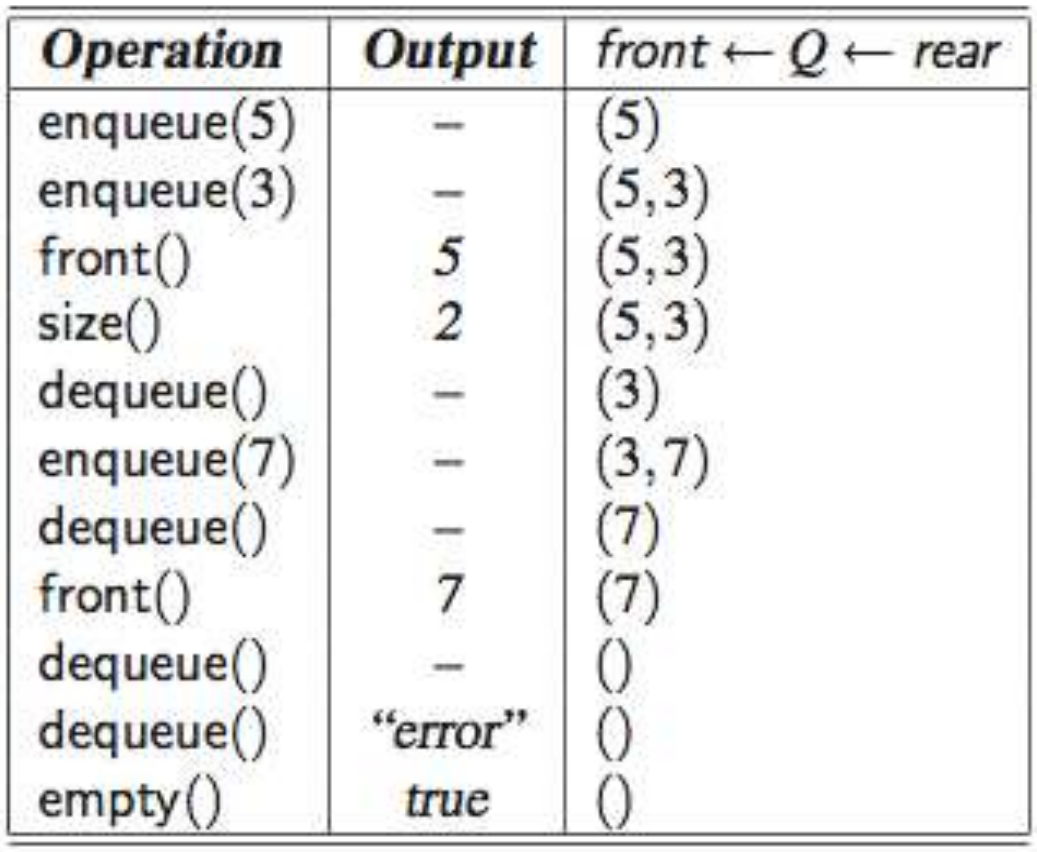
- Formally, the queue abstract data type defines a container that keeps elements in a sequence, where element access and deletion are restricted to the first element in the sequence, which is called the front of the queue, and element insertion is restricted to the end of the sequence, which is called the rear of the queue. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in first-out (FIFO) principle.

➢ The queue abstract data type (ADT) supports the following operations:

1. **enqueue(e)**: Insert element e at the rear of the queue.
2. **dequeue()**: Remove element at the front of the queue ; an error occurs if the queue is empty.
3. **front()**: Return, but do not remove, a reference to the front element in the queue; an error occurs if the queue is empty.
4. **size()**: Return the number of elements in the queue.
5. **empty()**: Return true if the queue is empty and false otherwise. We illustrate the operations in the queue ADT in the following example.

Example

Example **:** The following table shows a series of queue operations and their effects on an initially empty queue, Q, of integers.



Operation	**Output**	*front* ← *Q* ← *rear*
enqueue(5)	–	(5)
enqueue(3)	–	(5,3)
front()	5	(5,3)
size()	2	(5,3)
dequeue()	–	(3)
enqueue(7)	–	(3,7)
dequeue()	–	(7)
front()	7	(7)
dequeue()	–	()
dequeue()	“error”	()
empty()	true	()

Representation Of Queues In Memory

- Queues, like stack, can also be represented in memory using a linear array or a linear linked list. Even using a linear, we have two implementation viz. linear and circular. The representation of these two is same; the only difference is in their behavior.

1. **Representing a Queues Using An Array**

- To implement a queue, we need two variables, called front and rear that hold the index of first and last element of the queue and an array, named elem, to hold the elements of the queue. Assuming a queue whose elements are of type int and size is 10, a queue as in figure (Next slide).

Representing a Queues Using An Array

-1	-1	10											
Front	Rear	Size		0	1	2	3	4	5	6	7	8	9

Figure (2 a) representation of empty queue in the memory

0	4	10		5	7	12	8	9					
Front	Rear	Size		0	1	2	3	4	5	6	7	8	9

Figure (2 b) representation of a queue with 5 elements

2	4	10				12	8	9					
Front	Rear	Size		0	1	2	3	4	5	6	7	8	9

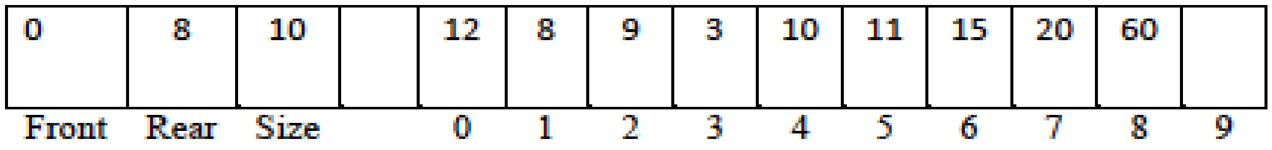
Figure (2 c) representation of a queue with 3 elements

2	9	10				12	8	9	3	10	11	15	20
Front	Rear	Size		0	1	2	3	4	5	6	7	8	9

Figure (2 d) representation of a queue with 8 elements

Representing a Queues Using An Array

- from the previous figure **(2d),** that is not possible to ***enqueue*** more elements as such, though two positions in the linear queue are vacant. To overcome this problem, the elements of the queue are moved forward so that the vacant position is shifted towards the rear end of the linear queue. After shifting, front and rear are adjusted properly, and then the element is enqueued in the linear queue as usual. For example, we want enqueue one more element, say 60, after making adjustments the queue will look as shown in figure (2e) below.



Front	Rear	Size		0	1	2	3	4	5	6	7	8	9
0	8	10		12	8	9	3	10	11	15	20	60	

Figure (2 e) representation of a queue after adjustment and value of 60 added

Algorithms for Insert and Delete Operations in Linear Queue

For Delete Operation

Delete-Queue(Queue, Front, Rear, Item)

Here, Queue is the place where data are stored. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Front element is assigned to Item.

```
1. If Front = M+1   then Print: Underflow and Return.     /*…Queue Empty
                                                    /*Where M initial value of front
2. Set  Item := Queue[Front]
3. Set Front := Front + 1
4. Return.
```

Algorithms for Insert and Delete Operations in Linear Queue

For insert Operation

insert-Queue(Queue, Front, Rear, Item)

Here, Queue is the place where data are stored. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Front element is assigned to Item.

```
1.   If Rear = n+1   then Print: full and Return.     /*…Queue overflow
                                                       /*Where n size of queue
2. Set  Item := Queue[rear]
3. Set Rear := Rear + 1
4. Return.
```

Example: Consider the following queue (linear queue

Rear = 4 and Front = 1 and N = 7

10	50	30	40			
1	2	3	4	5	6	7

(1) Insert 20. Now Rear = 5 and Front = 1

10	50	30	40	20		
1	2	3	4	5	6	7

(2) Delete Front Element. Now Rear = 5 and Front = 2

	50	30	40	20		
1	2	3	4	5	6	7

(3) Delete Front Element. Now Rear = 5 and Front = 3

		30	40	20		
1	2	3	4	5	6	7

(4) Insert 60. Now Rear = 6 and Front = 3

		30	40	20	60	
1	2	3	4	5	6	7

Drawback of Linear Queue

- Once the queue is full, even though few elements from the front are deleted and some occupied space is relieved, it is not possible to add anymore new elements, as the rear has already reached the Queue's rear most position.

Circular Queue

- This queue is not linear but circular.
- Its structure can be like the following figure
- In circular queue, once the Queue is full the "First" element of the Queue becomes the "Rear" most element, if and only if the "Front" has moved forward. otherwise it will again be a "Queue overflow" state.

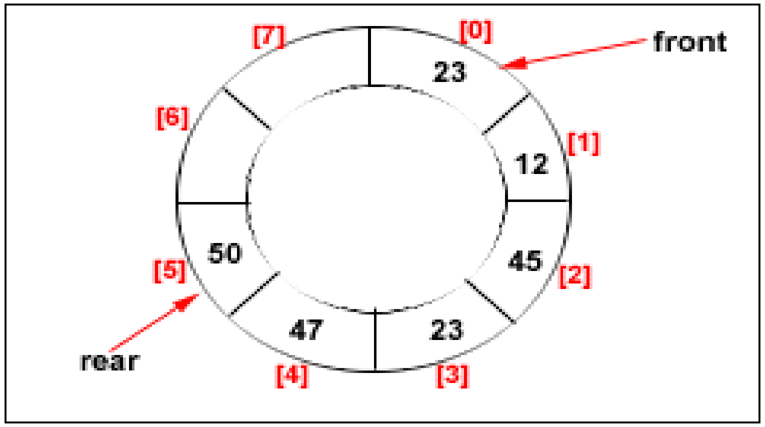


Figure: Circular Queue having
Rear = 5 and Front = 0

Algorithms for Insert and Delete Operations in Circular Queue

For Insert Operation

Insert-Circular-Q(CQueue, Rear, Front, N, Item)

Here, CQueue is a circular queue where to store data. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Here N is the maximum size of CQueue and finally, Item is the new item to be added. Initailly Rear = 0 and Front = 0.

```
1. If Front = 0 and Rear = 0 then Set Front := 1 and go to step 4.
2. If  Front =1 and Rear = N or Front = Rear + 1
           then Print: “Circular Queue Overflow” and Return.
3. If Rear = N  then Set Rear := 1 and go to step 5.
4. Set Rear := Rear + 1
5. Set CQueue [Rear] := Item.
6. Return
```

Algorithms for Insert and Delete Operations in Circular Queue

For Delete Operation

Delete-Circular-Q(CQueue, Front, Rear, Item)

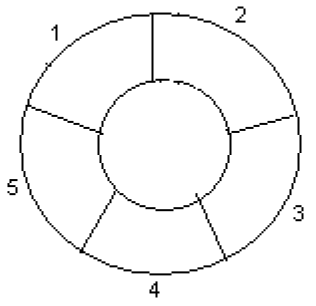
Here, CQueue is the place where data are stored. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Front element is assigned to Item. Initially, Front = 1.

```
1. If Front = 0 then
        Print: "Circular Queue Underflow" and Return.     /*..Delete without Insertion
2. Set Item := CQueue [Front]
3. If Front = N then Set Front = 1 and Return.
4. If Front = Rear then Set Front = 0 and Rear = 0 and Return.
5. Set Front := Front + 1
6. Return.
```

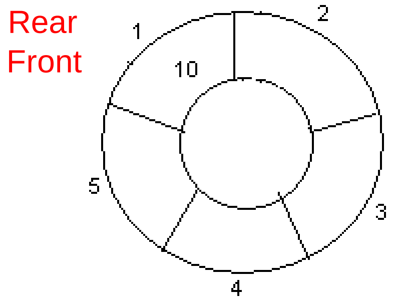
Algorithms for Insert and Delete Operations in Circular Queue

Example: Consider the following circular queue with N = 5.

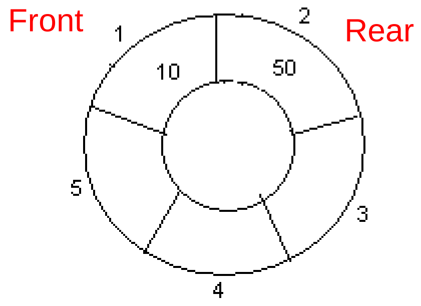
1. Initially, Rear = 0, Front = 0.



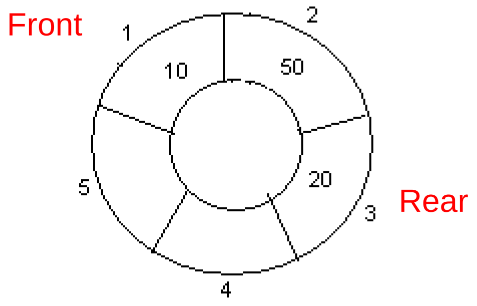
2. Insert 10, Rear = 1, Front = 1.



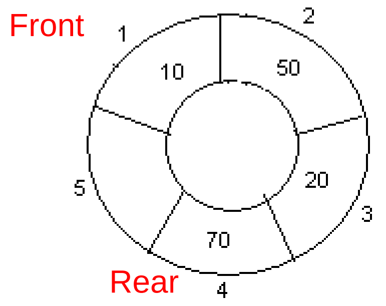
3. Insert 50, Rear = 2, Front = 1.



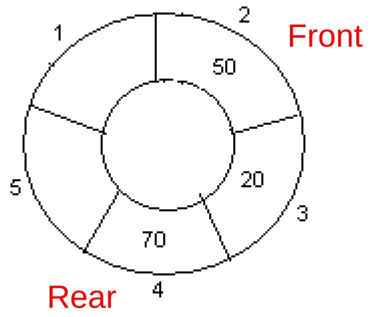
4. Insert 20, Rear = 3, Front = 1.



5. Insert 70, Rear = 4, Front = 1.

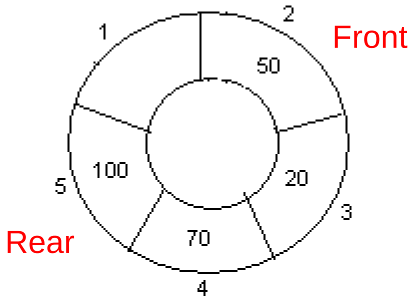


6. Delete front, Rear = 4, Front = 2.

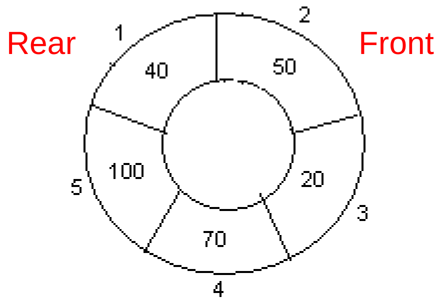


Algorithms for Insert and Delete Operations in Circular Queue

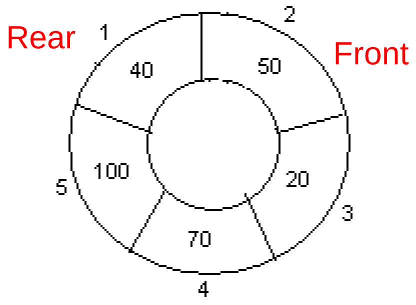
7. Insert 100, Rear = 5, Front = 2.



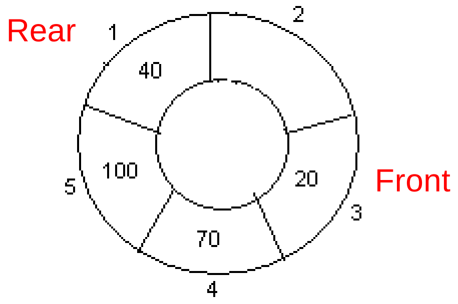
8. Insert 40, Rear = 1, Front = 2.



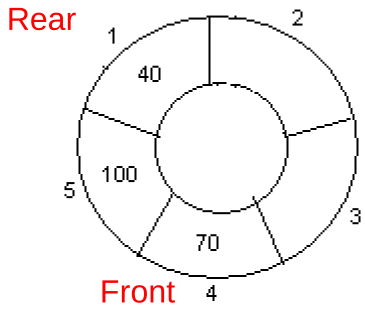
9. Insert 140, Rear = 1, Front = 2.
As Front = Rear + 1, so Queue overflow.



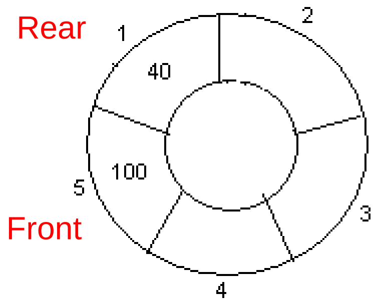
10. Delete front, Rear = 1, Front = 3.



11. Delete front, Rear = 1, Front = 4.



12. Delete front, Rear = 1, Front = 5.



THANK YOU