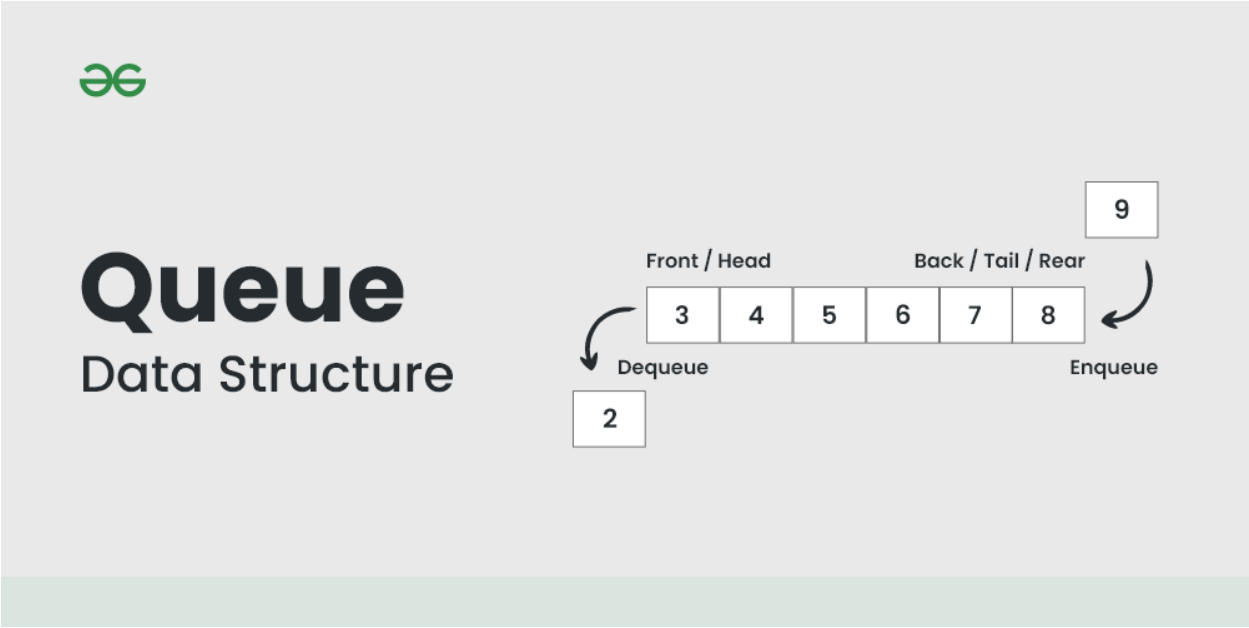
هياكل البيانات

Queue Data Structure

Chapter 2

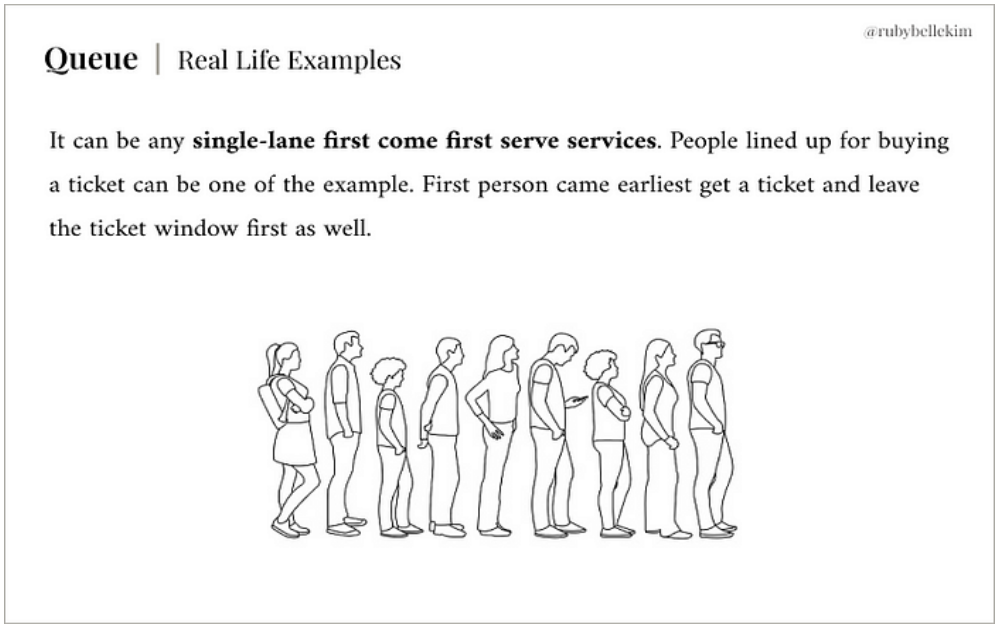


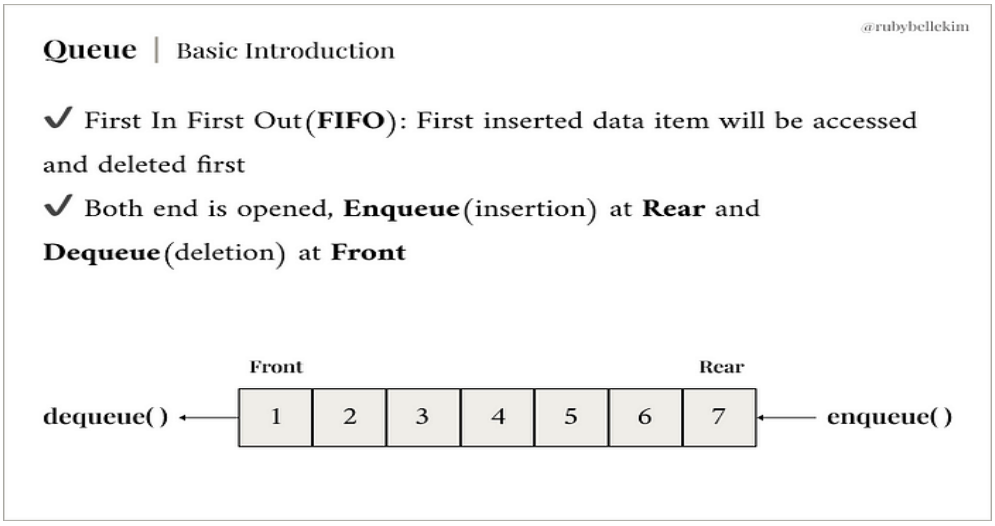
Prepared by Assist . Prof .
Imad Matti
AL-mamoon University College / Cyber Security Engineering Department

2024

Queues

A **queue** is a linear data structure that follows the First In, First Out (FIFO) principle. This means that the first element added to the queue is the first one to be removed.





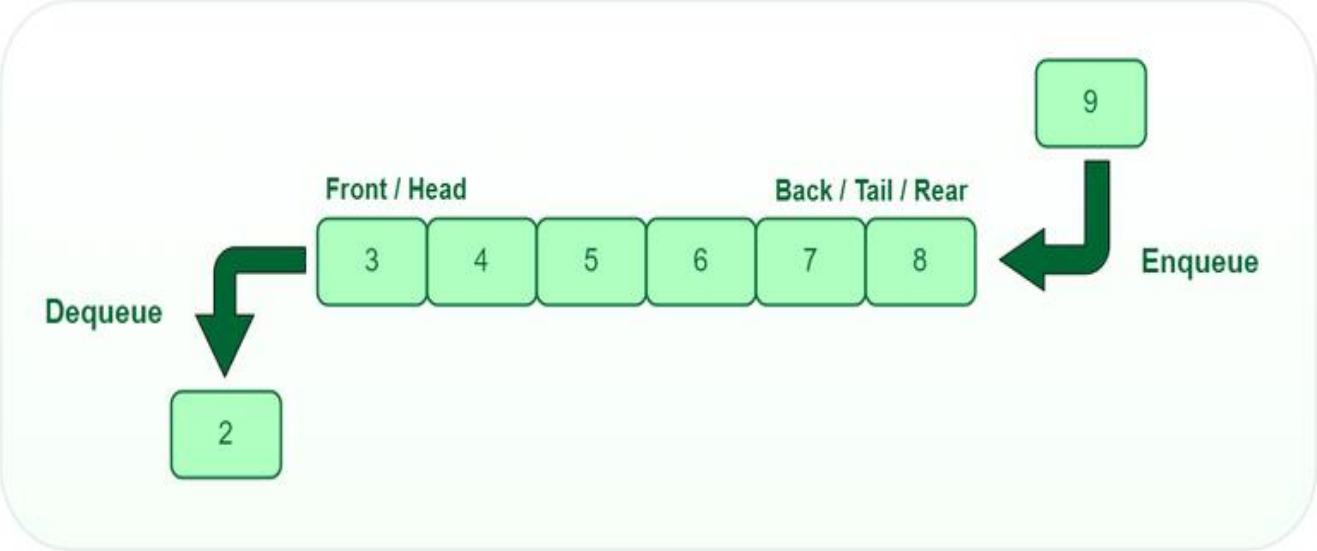
Basic Conception

Queue is a linear data structure that follows a particular order in which the operations are performed for storing data. The order is First In First Out **(FIFO)**.

- **enqueue()** : Insertion.
- **dequeue()** : Deletion.
- front : Accesses the data from the front end(for dequeuing).
- rear: Accesses the data from the rear end(for enqueuing).

Basic Operations for Queue in Data Structure

- **enqueue() –** Insertion of elements to the queue.
- **dequeue() –** Removal of elements from the queue.
- **peek() or front()-** Acquires the data element available at the front node of the queue without deleting it.
- **rear() –** This operation returns the element at the rear end without moving it.
- **isFull() –** Validates if the queue is full.
- **isEmpty() –** Checks if the queue is empty.
- **size():** This operation returns the size of the queue i.e. the total number of elements it contains.

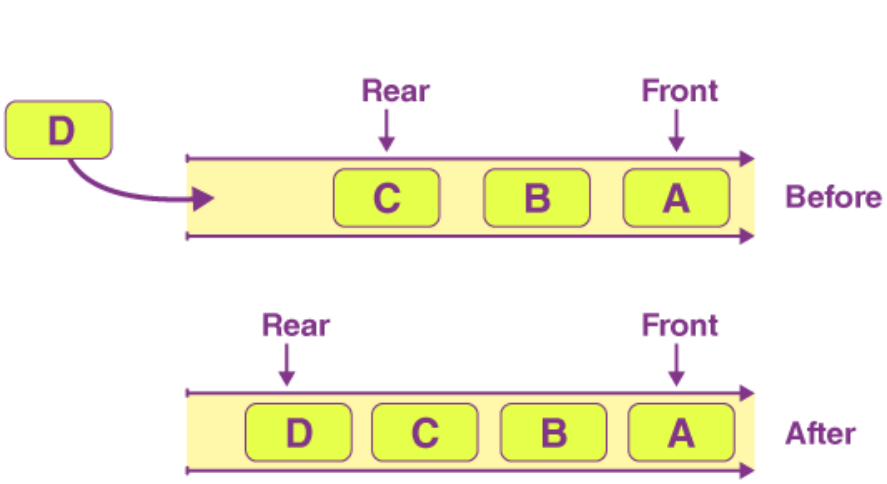


1. Insertion: enqueue()

The enqueue() Operation is used to insert an element at the back of a queue to the end of a queue or the rear end of the queue.

Algorithm

Step 1: Check if the queue is full.
Step 2: If the queue is full, produce an overflow error and exit.
Step 3: If the queue is not full, increment the rear pointer to point to the next space.
Step 4: Add a data element to the queue location, where the rear is pointing.



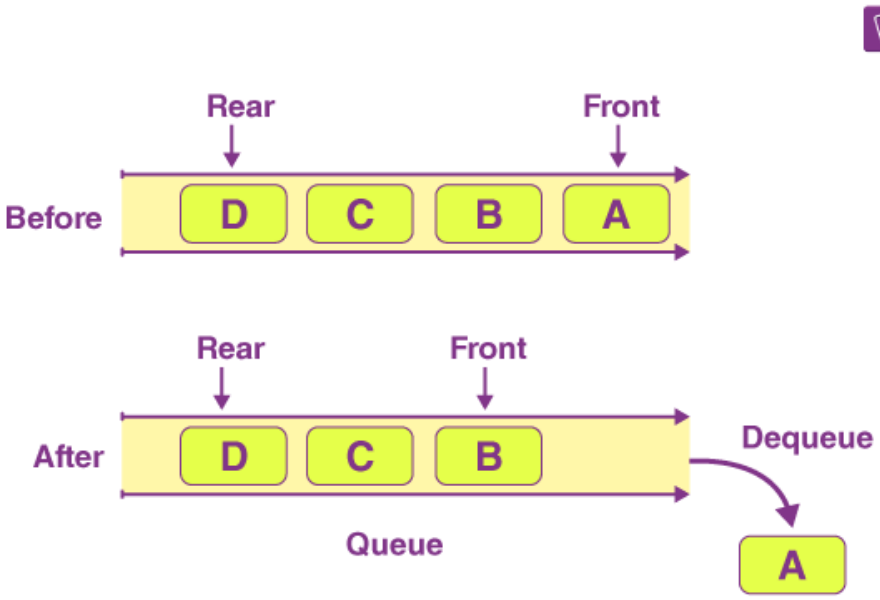
Queue Enqueue

2. Deletion: dequeue()

The dequeue() Operation is used to remove and return the element from the front of a queue.

Algorithm

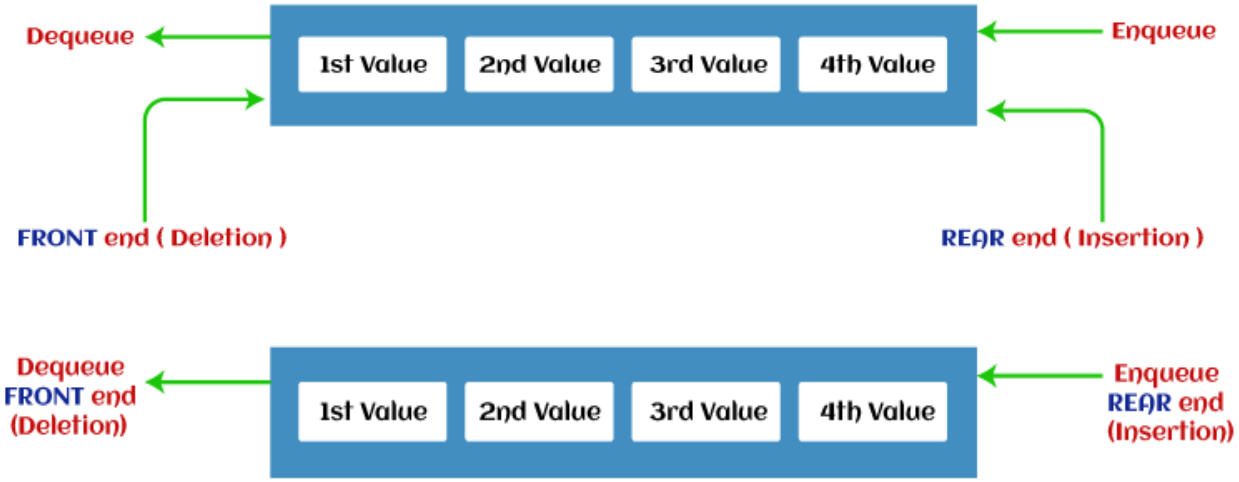
Step 1: Check **if** the queue is empty.
Step 2: If the queue is empty, print underflow and exit.
Step 3: If the queue is not empty, access the data where the front is pointing.
Step 4: Increment the front pointer to point to the next available data element.
Step 5: Set the front and rear as -1 **for** the last element.



Queue Dequeue

Simple Queue or Linear Queue

In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end. It strictly follows the FIFO rule.



3. peek()

The peek()The operation returns the value at the front end of the queue without removing it.

Algorithm

Step 1: Check **if** the Queue is empty.
Step 2: Return the element at the front of the queue
Step 3: End the process and exit.

4. isFull()

The isFull() operation is used to determine if the queue is full or not. A queue is said to be full if it has reached its maximum capacity and there is no more space to add new elements to it.

Algorithm

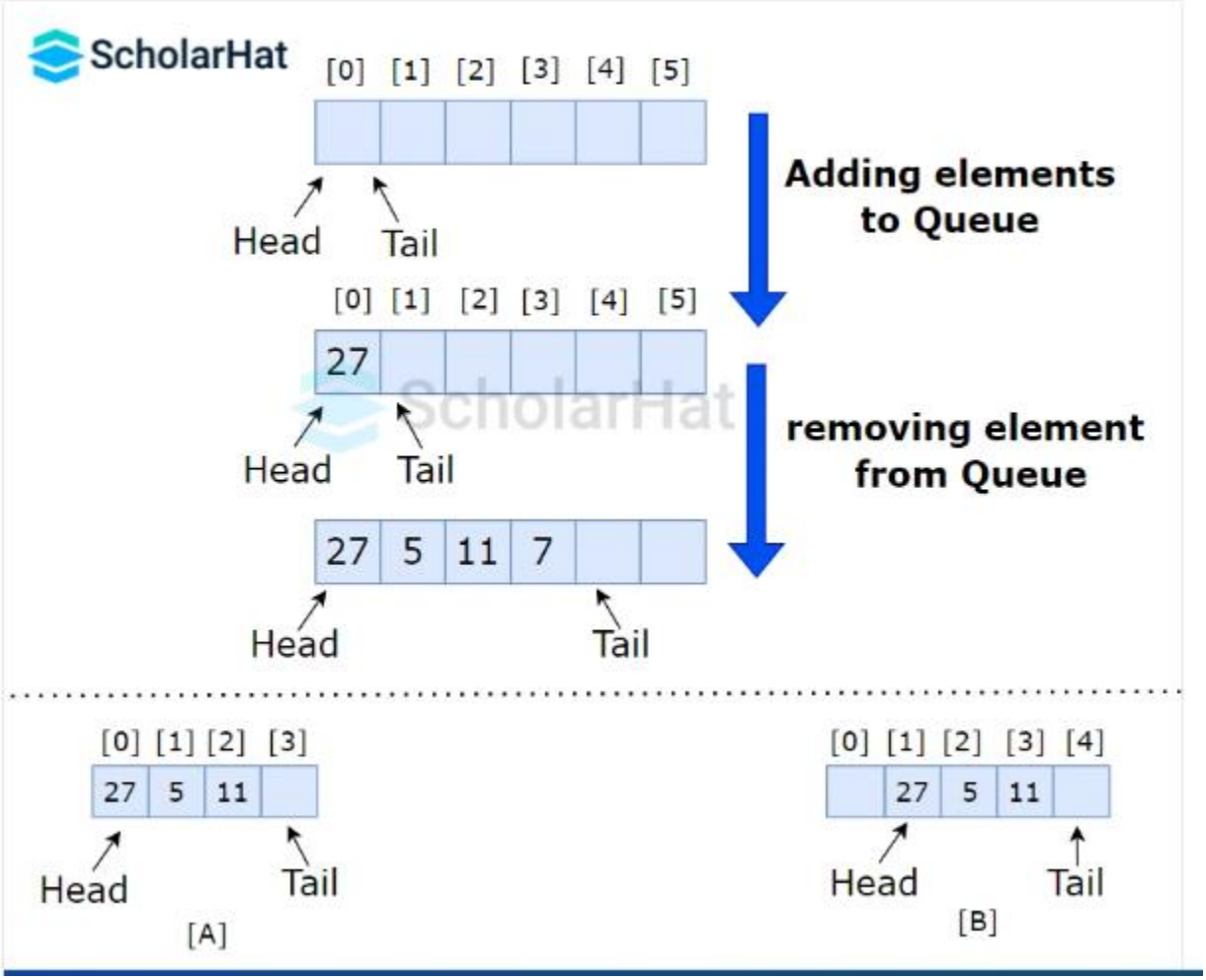
Step 1: If the count of queue elements equals the queue size, **return** true.
Step 2: Otherwise, **return** false
Step 3: End the process and exit.

5. isEmpty()

The isEmpty() operation is used to check if the queue is empty or not. It returns a boolean value, true when the queue is empty, otherwise false.

Algorithm

Step 1: If the count of queue elements equals zero, **return** true.
Step 2: Otherwise, **return** false
Step 3: End the process and exit.



Queue operations work as follows:

- Two pointers are there denoting two ends, FRONT and REAR.
- FRONT Tracks the first element of the queue.
- REAR Tracks the last element of the queue.
- Initially, set the value of FRONT and REAR to -1.
- Afterward, follow the above-given algorithms for the basic operations.

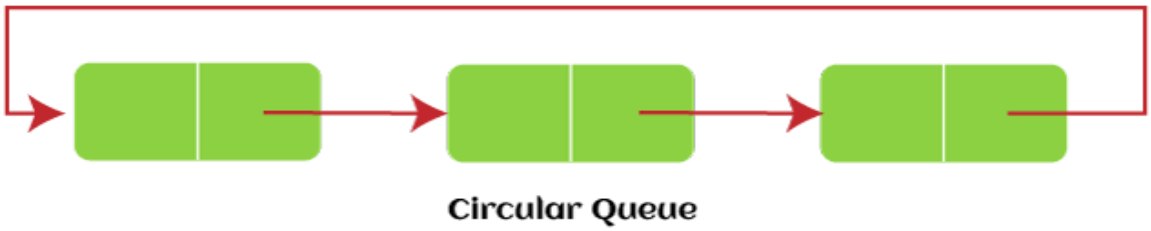
There are 4 types of queues in data structures:

1. Simple or Linear Queue

- As we mentioned above, a Simple queue follows the First-In-First-Out (FIFO) principle.
- In this, Items are taken out of the front and put in the back.
- It is necessary to remove outdated elements in order to add new ones.

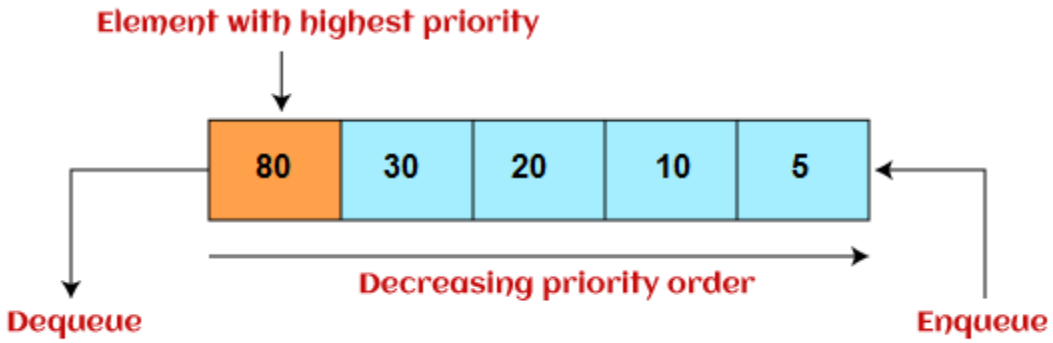
2. Circular Queue

- A **circular queue** is similar to a simple queue, but the last element is connected to the first element which creates a circular structure.
- This allows for efficient use of memory.
- It is also known as a **Ring Buffer**.



3. Priority Queue

- It is a special type of queue in which each element has a priority assigned to it.
- The element with the highest priority is removed first.
- This is useful in situations where certain elements need to be processed before others.



4. Dequeue (Double-Ended Queue)

- In this, the elements can be added or removed from both endsfront and rear of the queue.
- Dequeue allows insertion and deletion at both front and rear ends.
- It provides flexibility in managing data with operations from both ends.

Menu driven program to implement queue using array

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. #define maxsize 5
4. void insert();
5. void delete();
6. void display();
7. int front = -1, rear = -1;
8. int queue[maxsize];
9. void main ()
10.     {
11.         int choice;
12.         while(choice != 4)
13.         {
14.             printf("\n************************Main Menu*************************
    ***\n");
15.             printf("\n=========================================
    ======================\n");
16.             printf("\n1.insert an element\n2.Delete an element\n3.Display th
    e queue\n4.Exit\n");
17.             printf("\nEnter your choice ?");
18.             scanf("%d",&choice);
19.             switch(choice)
20.             {
21.                 case 1:
22.                 insert();
23.                 break;
24.                 case 2:
25.                 delete();
26.                 break;
27.                 case 3:
28.                 display();
29.                 break;
30.                 case 4:
31.                 exit(0);
32.                 break;
33.                 default:
```

```
34.                 printf("\nEnter valid choice??\n");
35.             }
36.         }
37.     }
38.     void insert()
39.     {
40.         int item;
41.         printf("\nEnter the element\n");
42.         scanf("\n%d",&item);
43.         if(rear == maxsize-1)
44.         {
45.             printf("\nOVERFLOW\n");
46.             return;
47.         }
48.         if(front == -1 && rear == -1)
49.         {
50.             front = 0;
51.             rear = 0;
52.         }
53.         else
54.         {
55.             rear = rear+1;
56.         }
57.         queue[rear] = item;
58.         printf("\nValue inserted ");
59.
60.     }
61.     void delete()
62.     {
63.         int item;
64.         if (front == -1 || front > rear)
65.         {
66.             printf("\nUNDERFLOW\n");
67.             return;
68.
69.         }
70.         else
71.         {
```

```
72.             item = queue[front];
73.             if(front == rear)
74.             {
75.                 front = -1;
76.                 rear = -1 ;
77.             }
78.             else
79.             {
80.                 front = front + 1;
81.             }
82.             printf("\nvalue deleted ");
83.         }
84.
85.
86.     }
87.
88.     void display()
89.     {
90.         int i;
91.         if(rear == -1)
92.         {
93.             printf("\nEmpty queue\n");
94.         }
95.         else
96.         {   printf("\nprinting values .....\n");
97.             for(i=front;i<=rear;i++)
98.             {
99.                 printf("\n%d\n",queue[i]);
100.            }
101.        }
102.    }
```

Output:

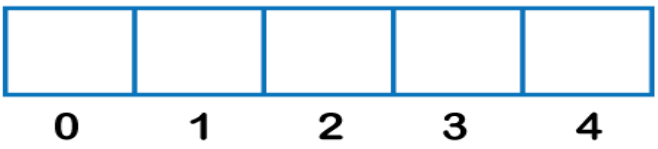
```
*************Main Menu**************

==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter the element
123

Value inserted

*************Main Menu**************

==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter the element
90

Value inserted

*************Main Menu**************
```

```
==================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?2

value deleted

*************Main Menu**************
==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3

printing values .....

90

*************Main Menu**************

==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?4
```

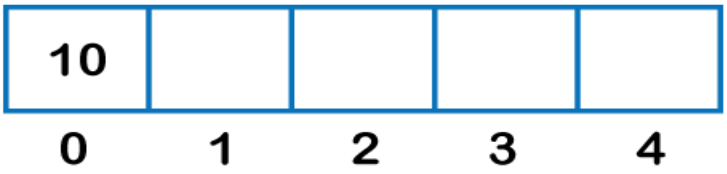
Circular Queue

What is a Circular Queue?

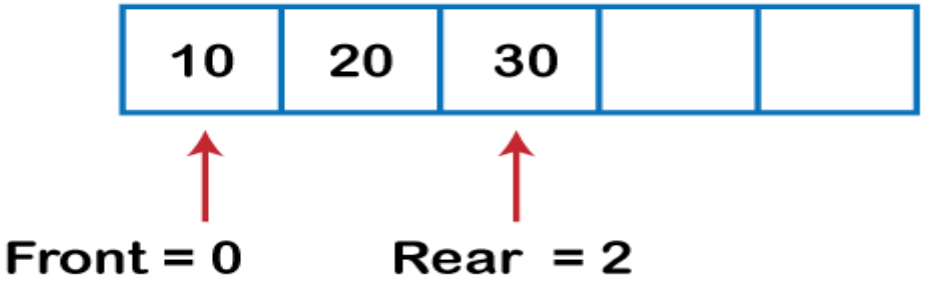
A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a ***Ring Buffer***.

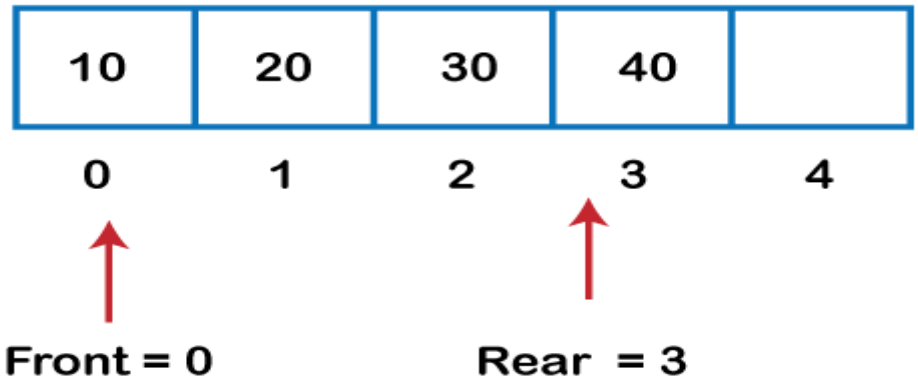


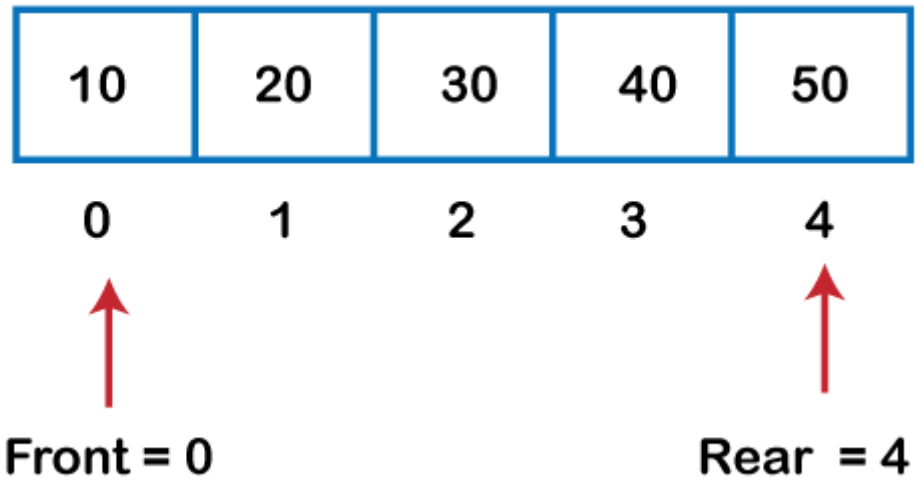
Front = -1
Rear = -1

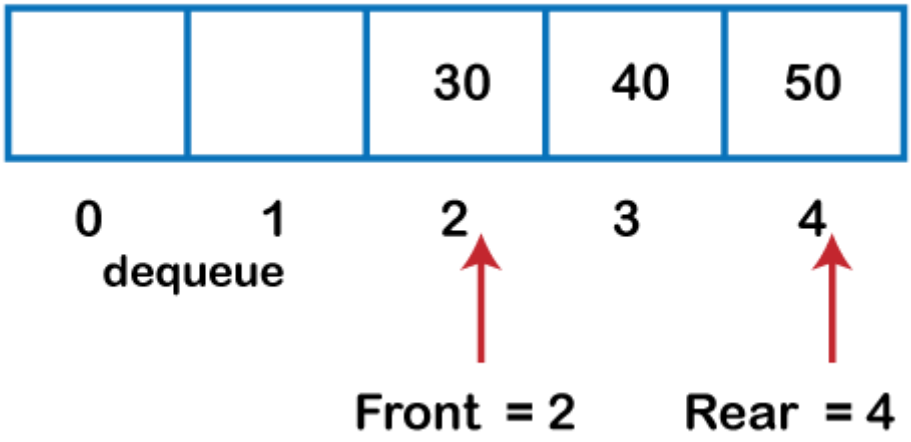


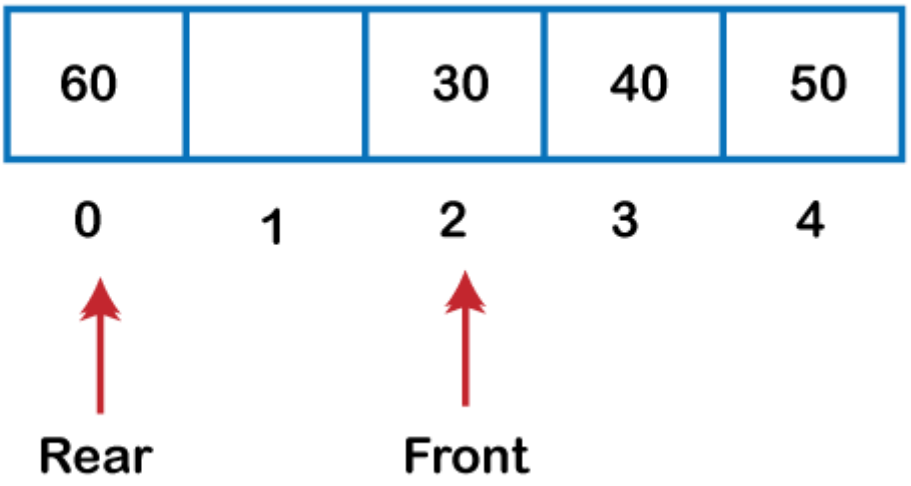
Front = 0
Rear = 0

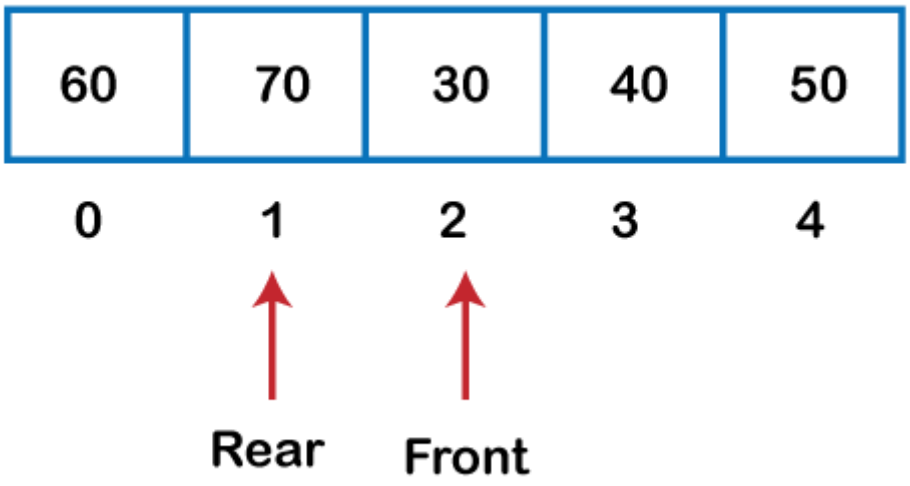

10
20
30
Front = 0
Rear = 2


10
20
30
40
0
1
2
3
4
Front = 0
Rear = 3


10
20
30
40
50
0
1
2
3
4
Front = 0
Rear = 4


30
40
50
0
1
2
3
4
dequeue
Front = 2
Rear = 4


60
30
40
50
0
1
2
3
4
Rear
Front


60
70
30
40
50
0
1
2
3
4
Rear
Front

Implementation of circular queue using Array

```
#include <stdio.h>

# define max 6
int queue[max];  // array declaration
int front=-1;
int rear=-1;
// function to insert an element in a circular queue
void enqueue(int element)
{
        if(front==-1 && rear==-1)   // condition to check queue is empty
        {
            front=0;
            rear=0;
            queue[rear]=element;
        }
        else if((rear+1)%max==front)  // condition to check queue is full
        {
            printf("Queue is overflow..");
        }
        else
        {
            rear=(rear+1)%max;       // rear is incremented
            queue[rear]=element;     // assigning a value to the queue at the rear position.
        }
    }

    // function to delete the element from the queue
    int dequeue()
    {
        if((front==-1) && (rear==-1))  // condition to check queue is empty
        {
            printf("\nQueue is underflow..");
```

```
33.          }
34.       else if(front==rear)
35.       {
36.          printf("\nThe dequeued element is %d", queue[front]);
37.          front=-1;
38.          rear=-1;
39.       }
40.       else
41.       {
42.           printf("\nThe dequeued element is %d", queue[front]);
43.          front=(front+1)%max;
44.       }
45.       }
46.       // function to display the elements of a queue
47.       void display()
48.       {
49.          int i=front;
50.          if(front==-1 && rear==-1)
51.          {
52.             printf("\n Queue is empty..");
53.          }
54.          else
55.          {
56.             printf("\nElements in a Queue are :");
57.             while(i<=rear)
58.             {
59.                printf("%d,", queue[i]);
60.                i=(i+1)%max;
61.             }
62.          }
63.       }
64.       int main()
65.       {
66.          int choice=1,x;   // variables declaration
67.
68.          while(choice<4 && choice!=0)   // while loop
69.          {
70.          printf("\n Press 1: Insert an element");
```

```
71.         printf("\nPress 2: Delete an element");
72.         printf("\nPress 3: Display the element");
73.         printf("\nEnter your choice");
74.         scanf("%d", &choice);
75.
76.         switch(choice)
77.         {
78.
79.            case 1:
80.
81.            printf("Enter the element which is to be inserted");
82.            scanf("%d", &x);
83.            enqueue(x);
84.            break;
85.            case 2:
86.            dequeue();
87.            break;
88.            case 3:
89.            display();
90.
91.         }}
92.         return 0;
93.      }
```

Output:

```
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
10

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
20

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
30

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
3

Elements in a Queue are :10,20,30,
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
2

The dequeued element is 10
```

Lab works

Select the appropriate option for each of the following questions

1. Queue follows the rule

   a) FIFO
   b) b) LIFO
   c) c) LILO
   d) d) FILO

2. The rear end of the queue is used

   a) To insert an element
   b) To remove an element
   c) Both a and b
   d) None of the above

3. When an element is inserted in a queue, the position of rear

   a) Increased
   b) decreased
   c) remains constant
   d) none of the above

4. If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they be removed?
   a) ABCD
   b) DCBA
   c) DCAB
   d) ABDC

5. A normal queue, if implemented using an array of size MAX_SIZE, gets full when?
   a) Rear = MAX_SIZE – 1
   b) Front = (rear + 1)mod MAX_SIZE
   c) Front = rear + 1
   d) Rear = front

6. Which of the following is not the type of queue?
   a) Ordinary queue
   b) Single ended queue
   c) Circular queue
   d) Priority queue

7. Let the following circular queue can accommodate maximum six elements with the following data

front = 2 rear = 4
queue = _____; L, M, N, ___, ___

What will happen after ADD O operation takes place?

a) front = 2 rear = 5
   queue = _____; L, M, N, O, ___

b) front = 3 rear = 5
   queue = L, M, N, O, ___

c) front = 3 rear = 4
   queue = _____; L, M, N, O, ___

d) front = 2 rear = 4
   queue = L, M, N, O, ___

8. A queue is?

a) FIFO (First In First Out) list
b) LIFO (Last In First Out) list.
c) Ordered array
d) linear tree

9. Which of the following conditions correctly check the underflow condition in a queue?
   1. Front=-1 2. Rear=1 3. Rear=size 4. None

10. In queue, deletion is happens at the ------- end.
    1. Rear 2. Front 3. Either rear or front 4. None

. Chapter 2 quizzes

1. Suppose a queue = 4, 5, 3, 7, 9, 6 where rear=4, and front= 6
   Draw the queue and perform 1.Dequeue 2. Enqueue 2

2. Suppose the initial state of Q1=1, 2, 3, 4 where the head=1, and Suppose the initial state of Q2 of four positions is empty as in the following figures.

Q1:

1	2	3	4

head

Q2:

Head

Perform only dequeue on Q1and enqueue on Q2 to reverse elements of Q1 into Q2.

The final state will be as follows:

the final state of Q1of four positions is empty and
the final state of Q2=4, 3, 2, 1 where the head=4.

Q1:

head

Q2:

4	3	2	1

Head