

UNIT- THREE

THE QUEUE

Topics to be Covered

- Introduction
- Queue as an ADT
- Primitive Operations in Queue
- Linear and Circular Queue
- Enqueue Dequeue and Priority Queue

Introduction: The Queue

- Queue is a linear list which has two ends, one for insertion of elements and other for deletion of elements.
- The first end is called Rear. Elements are inserted from rear
- The later end is called front. Elements are Deleted from the front end.
- It is also called FIFO list.
- Queue consist of homogenous collection of elements.

Example: people standing in a queue in bank. People standing in queue at gasoline center etc.

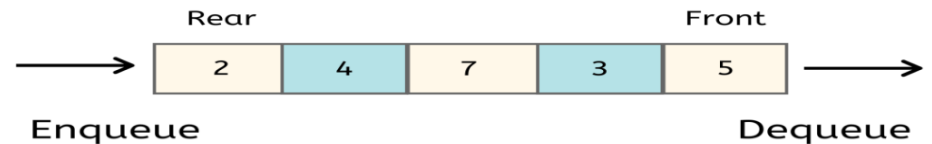
- Example – Queue in real world representation



Queue Representation

Queue

insertion and deletion
happens at different
ends



First in First out
(FIFO)

Empty Queue with 5 memory cells

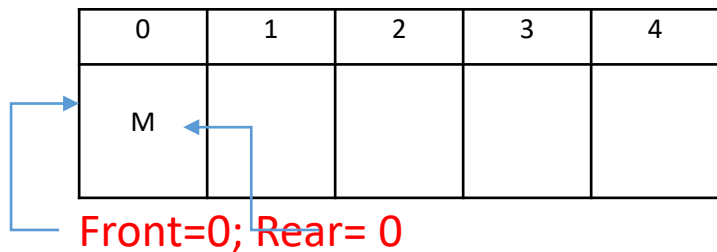
0	1	2	3	4

Front=-1; Rear= -1

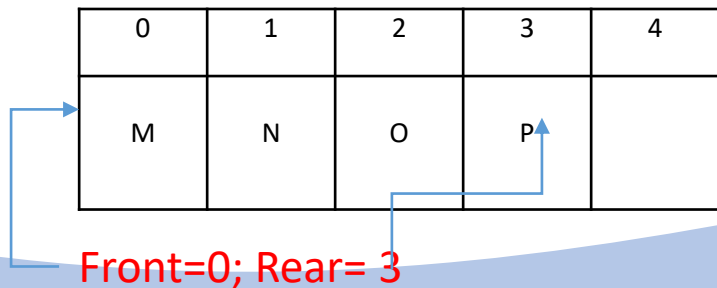
Deletion of Element



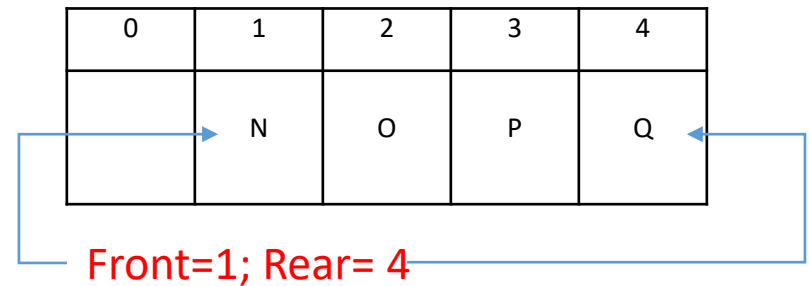
Insertion of first element 'M' in queue



Insertion of Element 'N', 'O', 'P'



Insertion of new element 'Q'



Queue as an ADT

- Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory.
- Basic operations associated with queues –
 - **enqueue()** – add (store) an item to the queue.
 - **dequeue()** – remove (access) an item from the queue.
- Functions to make the above-mentioned queue operation efficient. These are –
 - **peek()** – Gets the element at the front of the queue without removing it.
 - **isfull()** – Checks if the queue is full.
 - **isempty()** – Checks if the queue is empty.

isFull() Operation

```
bool isfull()  
{  
    if(rear == MAXSIZE - 1)  
        return true;  
    else  
        return false;  
}
```

isEmpty() Operation

```
bool isEmpty()  
{  
    if(front < 0 || front > rear)  
        return true;  
    else  
        return false;  
}
```


Enqueue() Operation

- Queues maintain two data pointers, **front** and **rear**. Therefore, its operations are comparatively difficult to implement than that of stacks.
- The following steps should be taken to enqueue (insert) data into a queue –
 - 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** pointer to point the next empty space.
 - Step 4** – Add data element to the queue location, where the rear is pointing.
 - Step 5** – return success.

Algorithm: Enqueue() Operation

Step 1: Start

Step 2: IF $REAR = MAX - 1$
Write OVERFLOW
Go to step 5
[END OF IF]

Step 3: IF $FRONT = -1$ and $REAR = -1$
SET $FRONT = REAR = 0$
ELSE
SET $REAR = REAR + 1$
[END OF IF]

Step 4: Set $QUEUE[REAR] = NEW_ITEM$

Step 5: Stop

Deque() Operation

- Accessing data from the queue is a process of two tasks – access the data where **front** is pointing and remove the data after access.
- The following steps are taken to perform **dequeue()** operation –
 - 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** pointer to point to the next available data element.
 - Step 5** – Return success.

Algorithm: Dequeue() Operation

Step 1: Start

Step 2: IF FRONT = -1 or FRONT > REAR

Write UNDERFLOW

ELSE IF FRONT = REAR

QUEUE[FRONT] = NULL

FRONT = -1

ELSE

SET VAL = QUEUE[FRONT]

SET FRONT = FRONT + 1

[END OF IF]

Step 3: STOP

Array Implementation of Queue

```
#include <stdio.h>
#include <conio.h>
#define MAX 100
int q[MAX + 1], front = -1, rear = -1;
void create();
void traverse(), insert(), delete();
void main ( )
{
    create ( );
    traverse ();
    insert ( );
    printf("\n After insert an element");
    traverse();
    delete ( );
    printf("\nAfter deletion");
    traverse ( );
    getch ( );
}

void create ( )
{
    char ch;
    front=0;
```

```
do
{
    rear++;
    printf ("\nInput element in queue:\n");
    scanf ("%d", & q[rear]);
    printf ("Press <Y/N> for more element");
    ch = getch ( );
}
while (ch=='Y');

void traverse ( )
{
    int i;
    printf ("\nelements in the Queue are:\n");
    for (i=front; i<=rear;i++)
        printf ("%d\n", q[i]);
}

void insert ( )
{
    int m;
    if (rear == MAX-1)
    {
        printf ("Queue is overflow \n");
        return;
    }
```

```
printf ("\nInput new element to insert\n");
scanf ("%d", &m);
rear++;
q[rear]=m;
}
```

```
void delete( )
{
    if (front==-1)
    {
        printf ("Queue is underflow\n");
        return;
    }
    if (front==rear)
    {
        q[front] = '\0';
        front = rear = -1;
    }
    else
    {
        q[front] = '\0';
        front++;
    }
}
```

Types of Queue

- There are four different types of queues:
- **Linear Queue**
- **Circular Queue**
- Priority Queue
- Double Ended Queue

Linear Queue

- Linear Queue has data elements that are arranged sequentially. The very basic principle of Queue is "**First in, First Out**" it means the element inserted first will get deleted first when the delete operation takes place.

The following are the features of linear queue:

1. Similar to a **stack**, the **Queue** is a list of items with similar data types.
2. Queues are arranged in a **FIFO (First In, First Out)** structure.
3. To **remove** a new element from the Queue, all the elements inserted before the new element must be removed.

Drawbacks of Linear Queue:

Suppose, a queue has maximum size, i.e, MAX=5, say 5 elements pushed and 2 elements popped(deleted).

- Now, if new 2 elements are to be added(even though 2 queue cells are free), **elements cannot be pushed. The queue is full, though it is empty.**

Circular Queue

- A circular queue is also a linear data structure like a normal queue that follows the FIFO principle but it does not end the queue; it connects the last position of the queue to the first position of the queue.
- It is also known as a ***Ring Buffer***.
- If we want to insert new elements at the beginning of the queue, we can insert it using the circular queue data structure.

Suppose Q is a query array of 6 elements. Push pop operation can be performed on circular fashion.

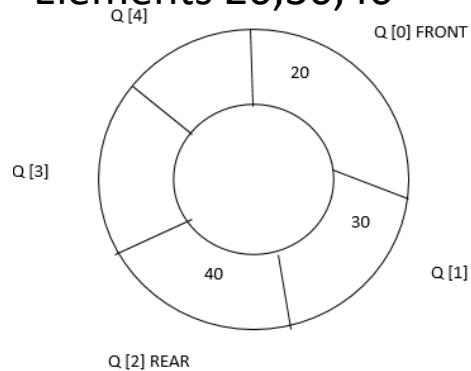
- Position of the Elements to be inserted will be calculated by:

$$\text{Rear} = (\text{Rear} + 1) \% \text{SIZE}$$

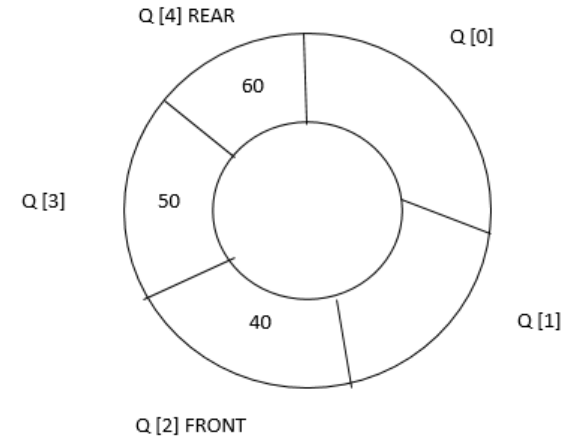
- For deletion, position of FRONT is calculated as:

$$\text{Front} = (\text{Front} + 1) \% \text{SIZE}$$

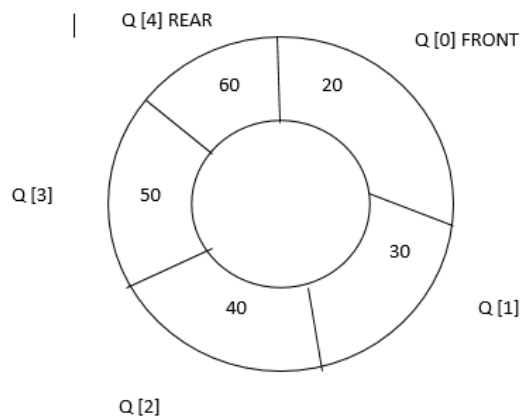
Circular queue with size: 5. Data Elements 20,30,40



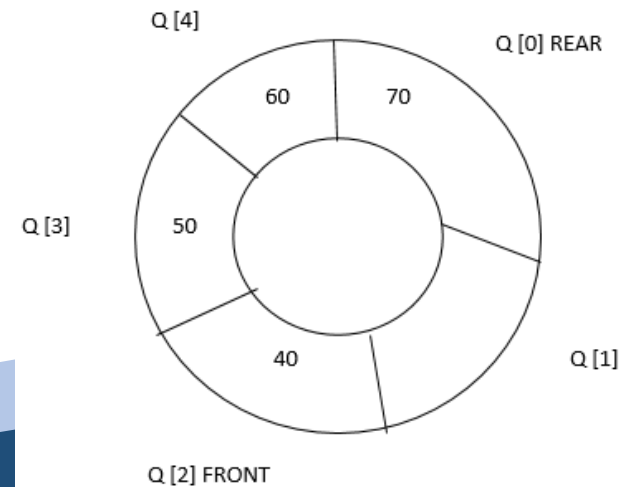
Deletion of Data Elements 20,30



Insertion of Data Elements 50,60



Insertion of Data Elements 70



Algorithm to Insert Element to Circular Queue

Step 1: START

STEP 2: IF $\text{FRONT} == 0 \ \&\& \ \text{REAR} == \text{MAX}-1 \ || \ \text{FRONT} == \text{REAR}+1$
WRITE " OVERFLOW" AND STOP

STEP 3: READ DATA

STEP 4: IF $\text{FRONT} == -1$
 $\text{FRONT} = 0; \text{REAR} = 0$
 ELSE IF $\text{REAR} == \text{MAX}-1$
 THEN $\text{REAR} = 0$
 ELSE
 $\text{REAR} = \text{REAR} + 1$

STEP 5: $\text{CQ}[\text{REAR}] = \text{DATA}$

STEP 6: STOP

Algorithm to delete in circular queue

STEP 1: START

STEP2: IF FRONT \equiv -1

 WRITE "UNDERFLOW" AND STOP

STEP 3: IF FRONT \equiv REAR THEN

 CQ[FRONT]=NULL

 FRONT=-1; REAR=-1

STEP 4: IF FRONT \equiv MAX-1

 CQ[FRONT]=NULL

 FRONT=0

 ELSE

 CQ[FRONT]=NULL

 FRONT=FRONT+1

STEP 5: STOP

Practice:

Consider the following circular queue capable of accommodating maximum 7 elements:

REAR= 3, FRONT=1

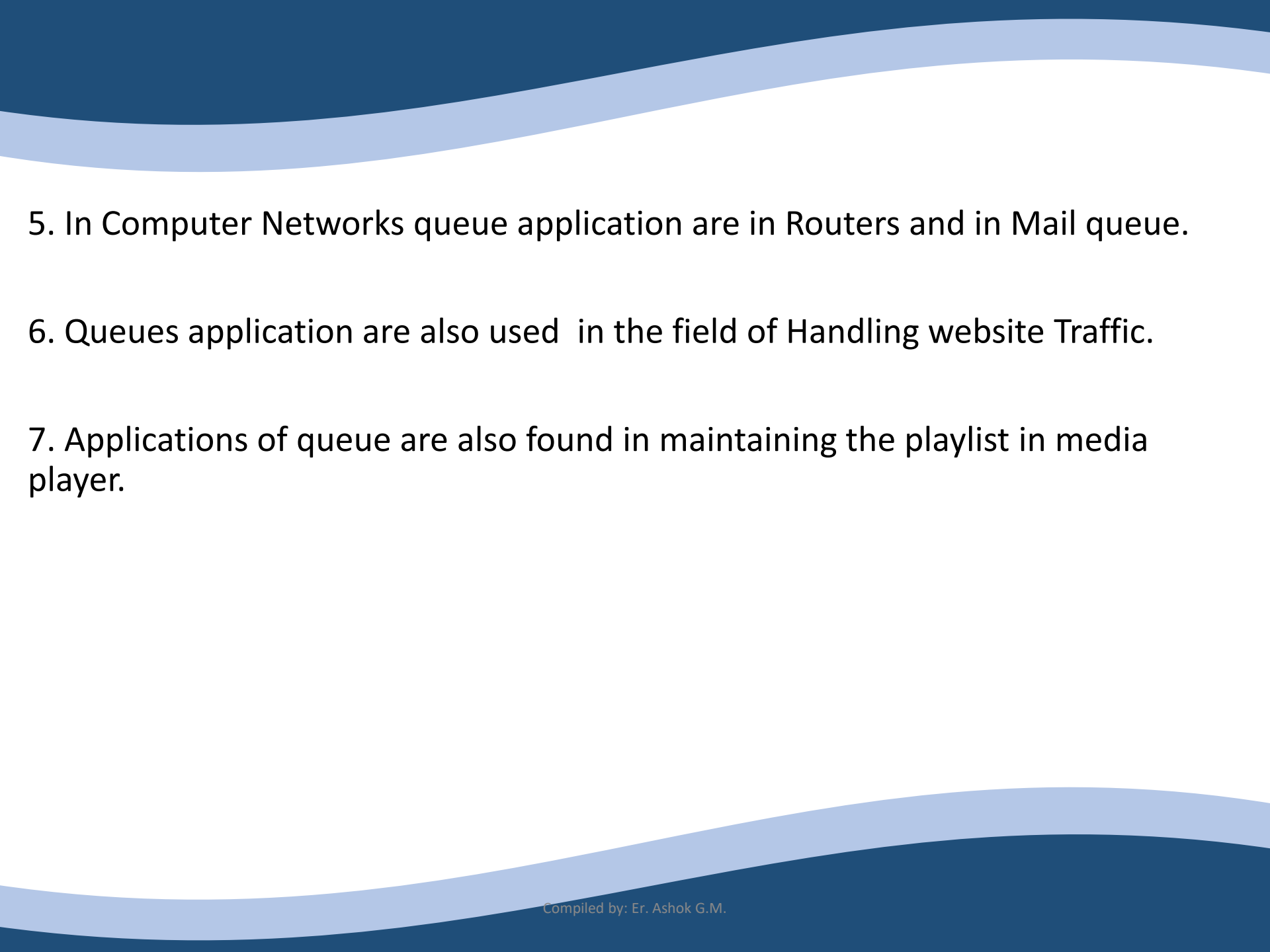
QUEUE: NULL,A,B,C,_,_,_

Describe the queue as the following operations take place.

- i) Add 'L' element
- ii) Add 'M' and 'N' element
- iii) Delete two elements
- iv) Add 'P', 'Q' , 'R'
- v) Delete one element

Application of queues

1. Job Scheduling: The jobs that are to be executed by the computer is scheduled to be executed one by one. There are many jobs like keyboard press, mouse click etc. in the system.
 - These jobs are brought in the main memory. These jobs are assigned to the processor one by one which is organized using a queue.
 - e.g. First In First Out and Round Robin processor scheduling in queues.
2. Multiprogramming: The various programs in the main memory are organized as queues and the queues formed are called 'Ready Queue'.
3. For Operation on Data Structure: Certain operations like **'tree traversal'** and **'Breadth first search uses queue for graph traversal'** involve the use of queues. These are some non-linear data structures. The sequence of traversal of inputs is set using queues.
4. Access to Shared Resources: In a networked system, the service is asked from a single resource simultaneously by various workstations or machines. The requests are generated in fractions of second, which is very fast. **This situation is maintained by implementing a queue in the device at the server.**

- 
5. In Computer Networks queue application are in Routers and in Mail queue.
 6. Queues application are also used in the field of Handling website Traffic.
 7. Applications of queue are also found in maintaining the playlist in media player.

Priority Queue

- A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.
- A priority queue is a collection of elements such that each element has been assigned an **implicit** or **explicit** priority.
- The priority of element is decided by its value is known as implicit priority and element with priority number assigned is known as explicit priority.
- The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.
- For example, suppose we have some values like 1, 3, 4, 8, 14, 22 inserted in a priority queue with an ordering imposed on the values is from least to the greatest. Therefore, the 1 number would be having the highest priority while 22 will be having the lowest priority.

Characteristics of a Priority queue

A priority queue is an extension of a queue that contains the following characteristics:

1. Every element in a priority queue has some priority associated with it.
2. An element with the higher priority will be deleted before the deletion of the lesser priority.
3. If two elements in a priority queue have the same priority, they will be arranged using the FIFO (first in first out) principle.

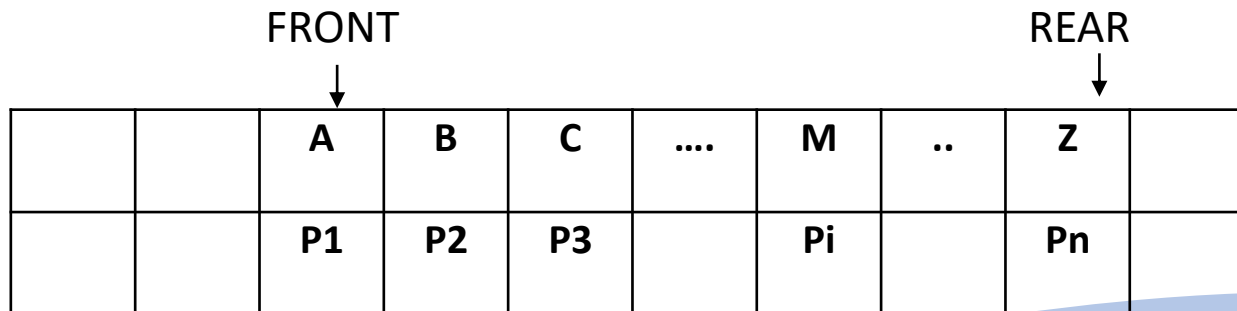


Fig: View of Priority Queue

Representation of the Priority Queue

There are various ways of maintaining a priority queue. These are:

- i. One way linked list
- ii. Multiple queues, one for each priority
- iii. Maximum or minimum heap

Representation of the Priority Queue: One way linked List

In this representation, each of node of linked list will have three field:

- a. An information field INFO
- b. A priority number PRN
- c. A Link LINK

Example:

	INFO	PRN	LINK
0	200	2	4
1	400	4	2
2	500	4	6
3	300	1	0
4	100	2	5
5	600	3	1
6	700	4	

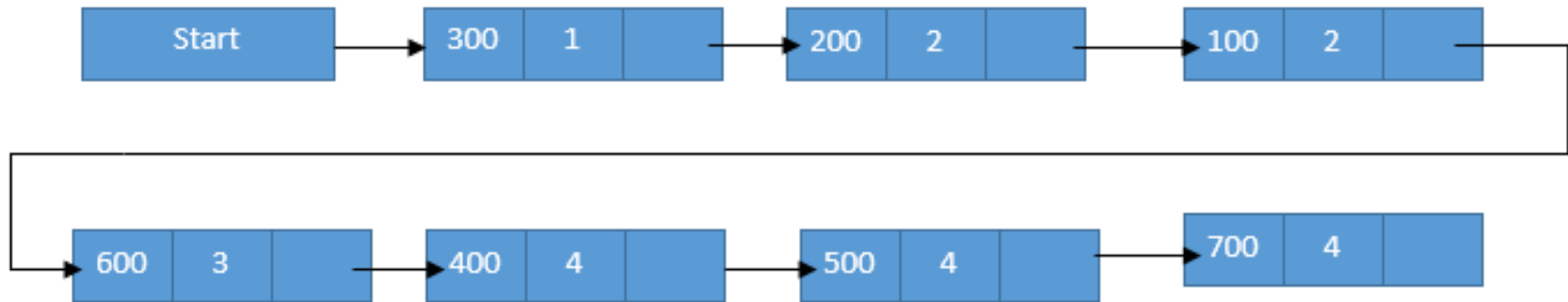
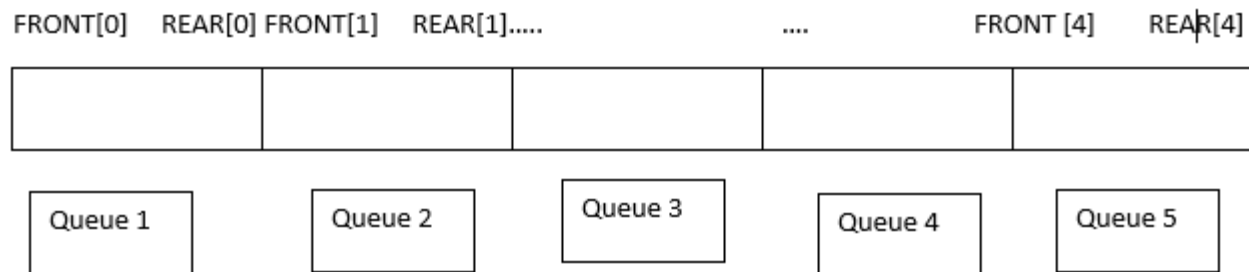


Figure: Schematic diagram of a priority queue with 7 elements

Representation of the Priority Queue: Multiple queues, one for each priority

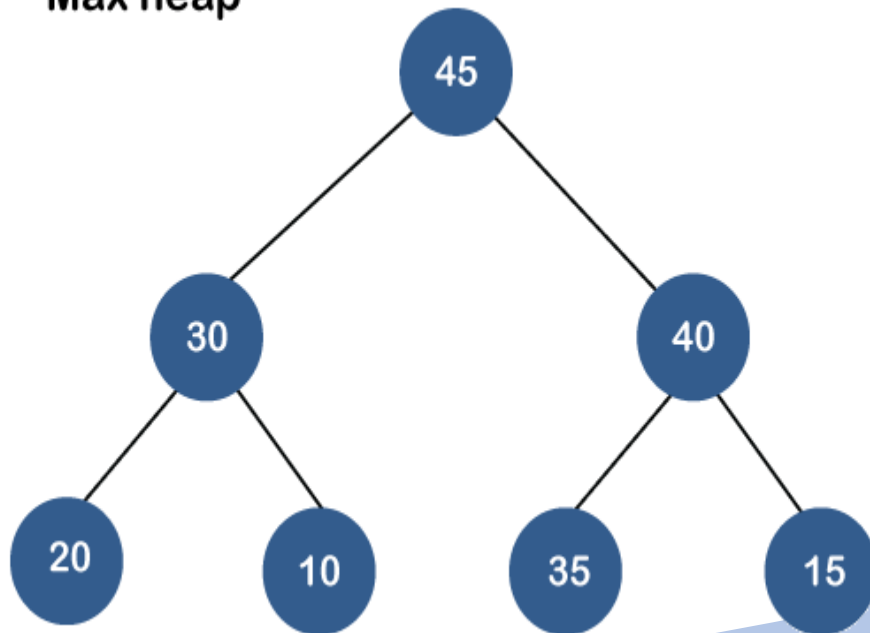
- Each queue is allocated a predefined space bounded by array indices
- In order to preserve multiple queues, there should be multiple FRONT and multiple REAR.
- One queue is maintained for each priority number.
- In order to process an element of the priority queue, element from the first non empty highest priority number queue is accessed.
- In order to add a new element to the priority queue, the element is inserted in an appropriate queue for a given priority.
- First queue has highest priority number elements, second queue has next higher priority number elements and so on.



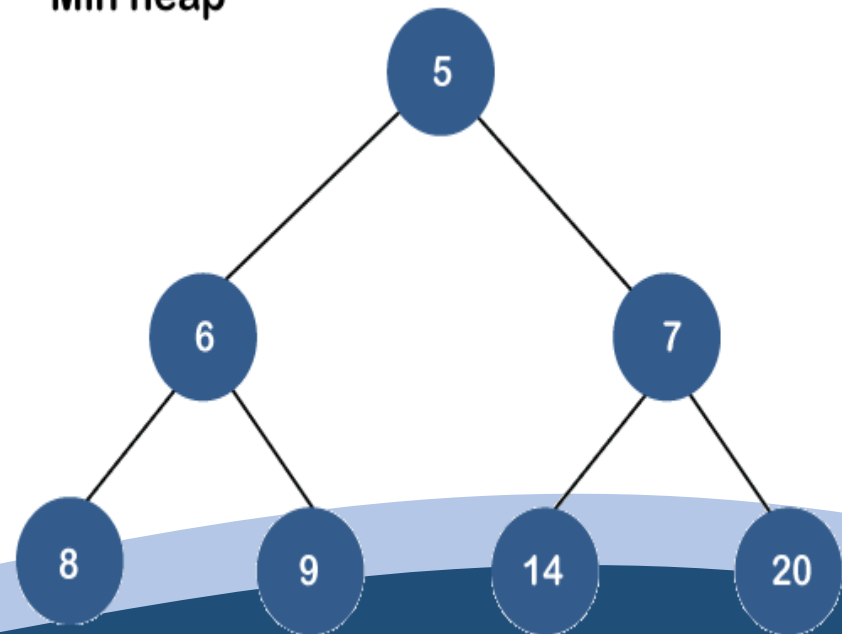
Representation of the Priority Queue: Maximum or minimum Heap

- A heap is a complete binary tree with additional property- the root node is either smallest or largest from its children.
- If the root node of the heap is smallest from its children, It is called min heap.
- If the root node of the heap is largest from its children, It is called max heap.

Max heap



Min heap



Work to be done

1. Write the difference between Stack and Queue
2. What is a circular queue? How is it different from ordinary Queue.
3. Write a C program to implement concept of Circular queue.

Any Query?