6.) A+B/C-D

= A+BC/-D

= ABC/+ -D

= ABC/+D-

7.) (A+B)/(C-D) = (AB+)/(C-D)

= AB+/CD-

= AB+CD-/

8.) (A+B)*C/D = (AB+)*C/D

= AB+C*/D

= AB+C*D/

9.) (A+B)*C/D+E^F/G

= (AB+)*C/D+E^F/G

= AB+C*/D+EF^/G

= AB+C*D/+EF^G/

= AB+C*D/EF^G/+

10) $A + [(B+C) + (D+E) * F] / G$.

12) $A + (B * C - (D / E \hat{} F) * G) * H$

$= A + (B * C - (D / E \hat{} F) * G) * H$

$= A + (B * C - D E F \hat{} / * G) * H$

$= A + (B * C - D E F \hat{} / G *) * H$

$= A + (B C * - D E F \hat{} / G *) * H$

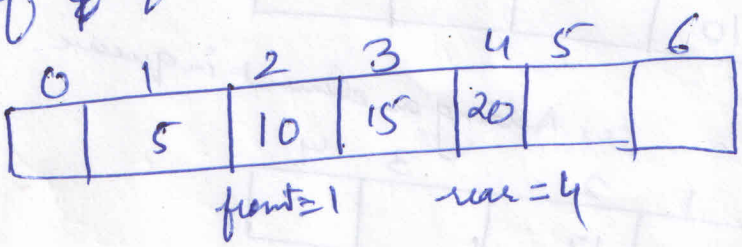
$= A + B C * D E F \hat{} / G * - * H$

$= A + B C * D E F \hat{} / G * - H *$

$= A B C * D E F \hat{} / G * - H * +$

Array Implementation of Queue

- As stack, Queue is also a collection of same type of elements. Therefor we can use array for implementing the queue.
- As, to add an item in queue at the rear end & delete the item in the queue at the front. Therefore we take 2 variables: rear (keeps the status of last added item in queue) & front (keeps the status of 1st item of queue.
- There may be possibly a condition when there is no place for adding elements in queue, this condition is overflow. so first check the value of rear with the size of array.
- The second possibility arises when there is no element for deleting from queue. If there is no element in queue then value of front & rear will be -1 or front will be greater than rear. So check the condition before deleting the element of queue.



Add operation in queue

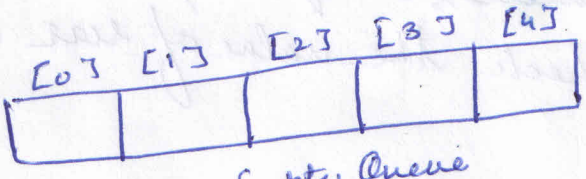
- For adding an element in queue, first we check the condition of overflow then add the elements:

```
if(rear == Maxsize-1)
    printf("Queue Overflow\n");
else
{   if(front==-1)
        front=0;
    rear=rear+1
    queue_a[rear]=item;
}
```

Queue

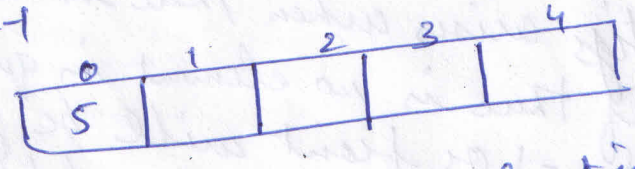
- Example of queue in daily life as queue of people, queue of cars, etc.
- Queue is an ordered list of elements in which we can add elements only at one end, called the rear of the queue & delete elements only at the other end called the front of the queue.
- In the queue of people or cars that comes first in the queue will be out first. So queue is called FIFO data structure.

- queue_arr

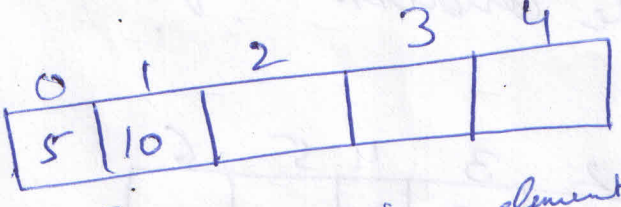


(a) Empty Queue

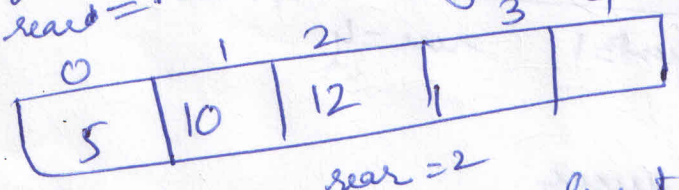
front=-1
rear =-1



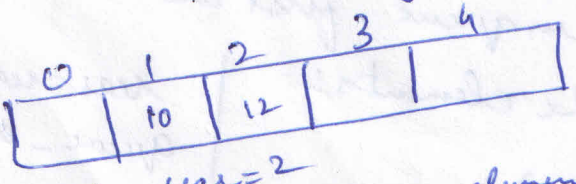
front=0
rear =0
(b) Adding an element in queue.



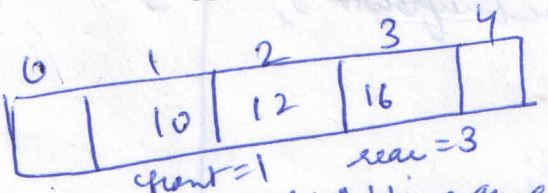
front=0
rear=1.
(c) Adding an element in queue.



front=0 rear=2
(d) Adding an element in queue



front=1, rear=2
(e) Deleting an element from queue.



front=1 rear=3
(f) Adding an element in queue.

Deletion operation in queue

- For deleting the element of queue, first check the condition of underflow then delete the element as:

```
if (front == -1 || front > rear)
  {
    printf ("Queue Underflow \n");
      return;
  }
else
  {
    printf ("Element deleted from queue %d \n", Queue_a[front]);
        front = front + 1;
  }

Display()
{
  int i;
  if ( front == -1)
        printf ("Queue is empty \n");

  else
    {
        printf ("Queue is : \n");
        for (i = front ; i <= rear ; i++)
          printf ("%d ", a[i]);
          printf ("\n");
    }
}
```

Linked list representation of Stack.

```
struct node {
        int info;
        struct node *link;
      };

push()
{
  struct node *tmp;
  int item;
  tmp = (struct node *)malloc (sizeof (struct node));
  printf ("Input the new value to be pushed on the stack:");
  scanf ("%d", &item);
  tmp->info = item;
  tmp->link = top;
  top = tmp;
}

pop()
{
  struct node *tmp;
  if (top == NULL)
      printf ("stack is empty\n");
  else
      { tmp = top;
        printf ("Popped item is %d\n", tmp->info);
        top = top->link;
        free (tmp);
      }
}
```

Linked List representation of Queue

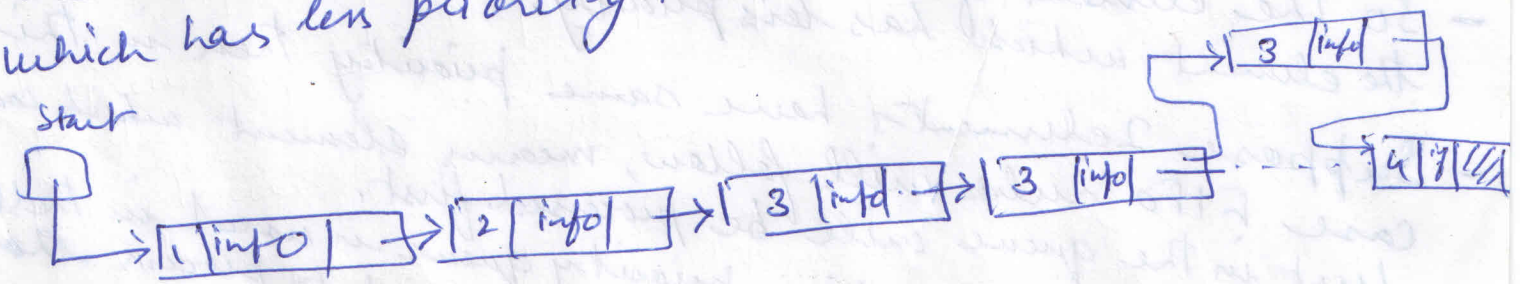
```
insert()
{
  struct node *tmp;
  int item;
  tmp=(struct node *) malloc(sizeof(struct node));
  printf("I/P the element for adding in queue:");
  scanf("%d", &item);
  tmp->info = item;
  tmp->link = NULL;
  if(front == NULL)
      front = tmp;
  else
      rear->link = tmp;
  rear = tmp;
}

del()
{
  struct node *tmp;
  if(front == NULL)
      printf("Queue Underflow\n");
  else
  {
      tmp = front;
      printf("Deleted element is %d\n", tmp->info);
      front = front->link;
      free(tmp);
  }
}
```

Operation in priority queue

1. Add
2. Delete

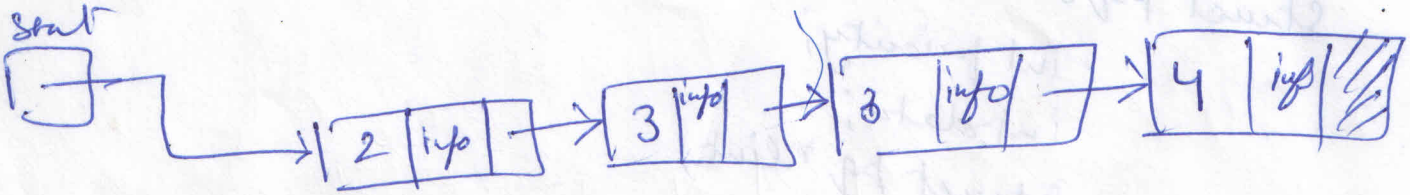
Add operation in Priority Queue

- Add operation in priority queue is same as the insert operation in sorted linked list.
- We insert the new element on the basis of priority of element.
- The new element will be inserted before the element which has less priority than new element.



Delete operation in Priority Queue

- Delete operation will be the deletion of 1st element of list as it has more priority than other elements of list.



Dequeue

- Queue is an ordered list of elements in which we can add the element at one end & called rear & delete the element only at other end called front of queue.
- But in dequeue, also called double ended queue, as the name implies we can add or delete the element from both sides.

Priority Queue

- Queue is an ordered list of elements in which we can add the element only at one end called rear of the queue & delete the element only at the other end called front of the queue.
- But in priority queue every element of queue has some priority & based on that priority queue every element of queue has same priority & based on that priority it will be processed.
- So the element of more priority will be processed before the element which has less priority.
- Suppose 2 elements have same priority then in this case FIFO rule will follow, means element which comes first in the queue will be processed first.
- In computer implementation, priority queue is used in multiple scheduling algo., in which CPU has need to process those processes at which have more priority.
- Linked list implementation of PQ.

```
Struct pq{
        int priority;
        int data;
        Struct pq *link;
}
```