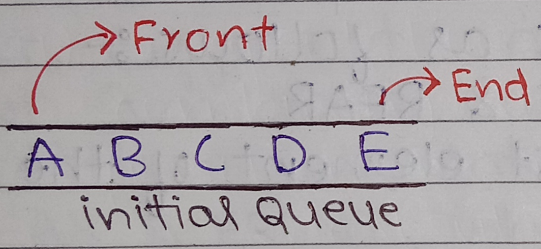
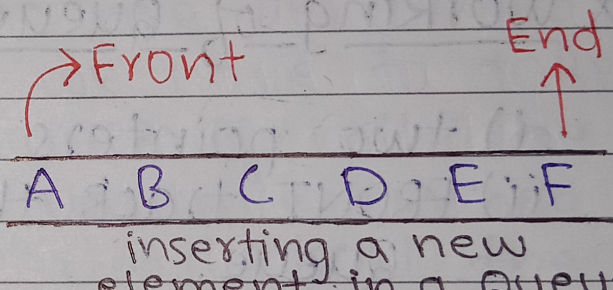
Queue

Queue is a linear list in which elements can be inserted only at one end called Rear of the Queue & delete only at the other end called Front of the Queue. The behaviour of Queue is First In First Out, so it is also called FIFO Data Struct -ure.

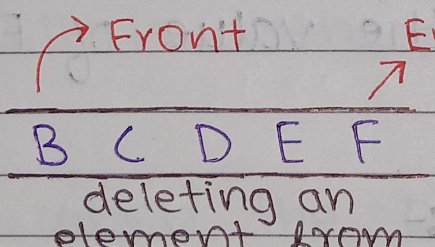
New element is inserted at the end called Rear & the deletion is done at the other end called Front.



initial Queue



inserting a new element in a Queue



deleting an element from Queue

In Queue, insertion operation is known as Enqueue & deletion operation is known as Dequeue. If insert operation is attempted & there is not enough space in the Queue, then this situation is called overflow & new element can't be inserted. If Queue is empty & delete operation is attempted, then this is called underflow & item can't be deleted.

* Basic Operations of Queue :-

A queue is an object (an abstract data structure - ADT) that allows the following Operations :-

i) Enqueue : Add an element to the end of the queue.
ii) Dequeue : Remove an element from the front of the queue.
iii) isEmpty : Check if the queue is empty.
iv) isFull : check if the queue is full.
v) Peek : Get the value of the front of the Queue without removing it.

* working of Queue :- Queue operations works as follows :-

i) two pointers FRONT & REAR
ii) FRONT track the first element of the Queue.
iii) REAR track the last element of the Queue.
iv) Initially, set the value of FRONT & REAR to -1.

enqueue operation :-

a) Check if the queue is full
b) For the first element, set the value of FRONT to 0.
c) Increase the rear index by 1.
d) Add the new element in the position pointer to by REAR.

dequeue operation :-

a) Check if the queue is empty.
b) return the value pointed by FRONT.
c) increase the front index by 1.
d) For the last element, reset the values of FRONT and REAR to -1.

* implementing Queue using Array

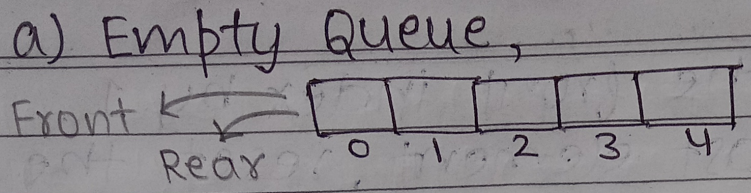
In stack, both operations were performed at the same end, so we had to take only one variable, i.e., top, but here in Queue, both operations are performed at different ends so we have to take two variables to keep track of both the ends. we will take the variable named REAR & FRONT, where rear will hold the index of last added item in the Queue & front will hold the index of first item of Queue.

i) initially when the Queue is empty, the value of both front & rear is -1.
ii) For insertion, the value of rear is incremented by 1 & the element is inserted at the new rear position.
iii) For deletion, the element at front is deleted & the value of front is incremented by 1.
iv) when insertion is done in an initially empty queue, i.e., if the value of front

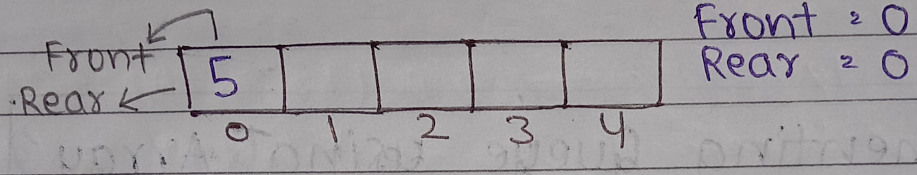
is -1, the value of front is made 0.

Example of Queue using array :- (4)

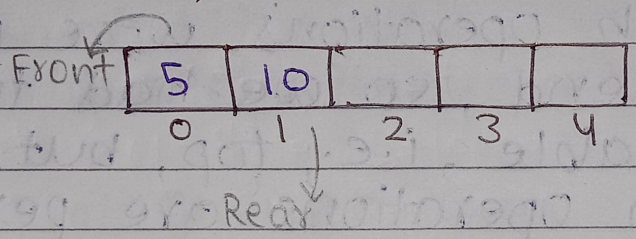
a) Empty Queue,



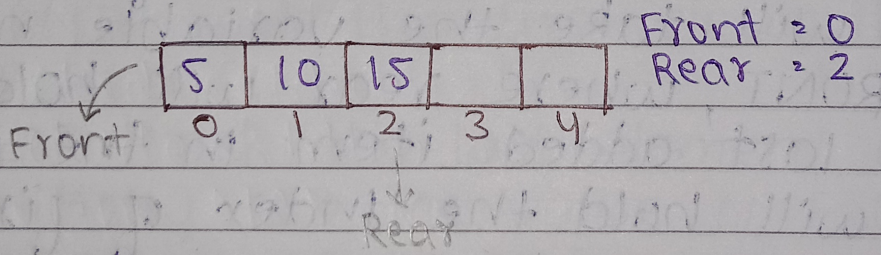
b) Insert 5,



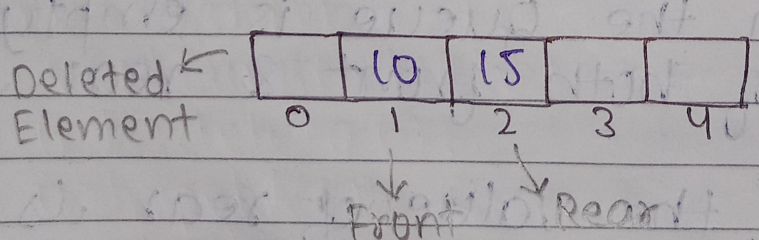
c) Insert 10,



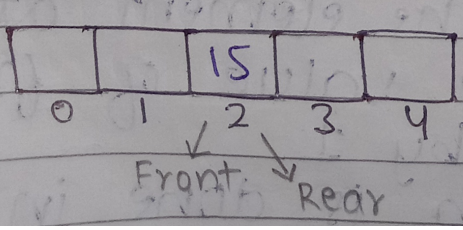
d) Insert 15,



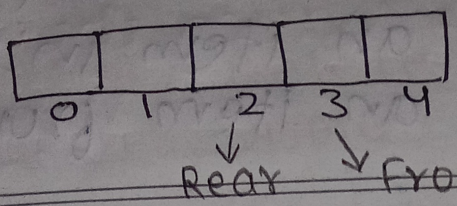
e) Delete Element,



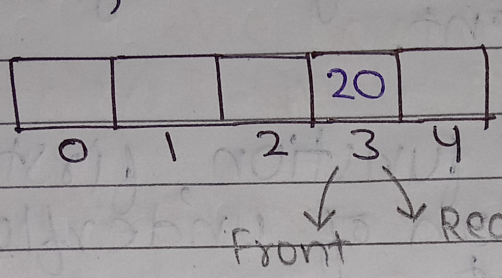
f) Delete Element,



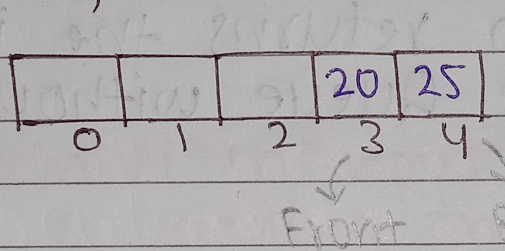
g) Delete Element,



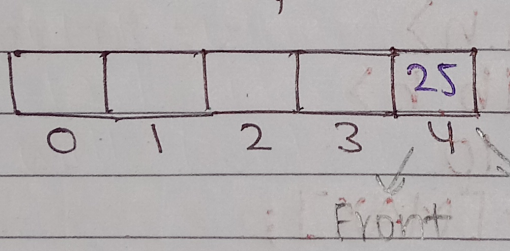
h) Insert 20,



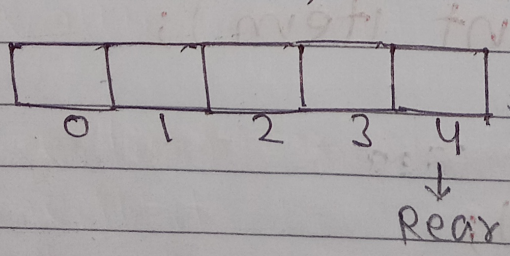
i) Insert 25,



j) Delete Element,



k) Delete Element,



There are two functions, insert() & delete() insert() will insert an item in the Queue & delete() will delete an item from the Queue.

Inside the insert() function, first we will check the condition of overflow & then insert the element.

Inside the delete() function, first we will check the condition of underflow & then delete the element.

The peek() function returns the item at the front of the Queue without removing it.

* Program of Queue :-

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int queue_arr[MAX];
int rear = -1;
int front = -1;
void insert(int item);
int delete();
int peek();
void display();
int isFull();
int isEmpty();
int main()
{
    int choice, item;
    while(1)
```

```
{
    printf("\n1. Insert");
    printf("\n2. Delete");
    printf("\n3. Display element at the
                front");
    printf("\n4. Display all element of
                the Queue");
    printf("\n5. Quit");

    printf("\nEnter your choice: ");
    scanf("%d", &choice);
    switch(choice)
{
    case 1:
        printf("\nInput the element of
                adding in Queue");
        scanf("%d", &item);
        insert(item);
        break;

    case 2:
        item = delete();
        printf("\n Deleted item is: %d",
                                item);
        break;
    case 3:
        printf("\nElement at front is:
                                %d", peek());
        break;
    case 4:
        display();
        break;
    case 5:
        exit(1);
```

```
        default:
            printf("\n wrong choice");
    }
  }
}

void insert (int item)
{
  if (isFull())
  {
    printf("\n Queue Overflow.");
    return;
  }
  if (front == -1)
  {
    front = 0;
  }
  rear = rear + 1;
  queue_arr[rear] = item;
}

int delete()
{
  int item;
  if (isEmpty())
  {
    printf("\n Queue underflow.");
    exit(1);
  }
  item = queue_arr[front];
  front = front + 1;
  return item;
}
```

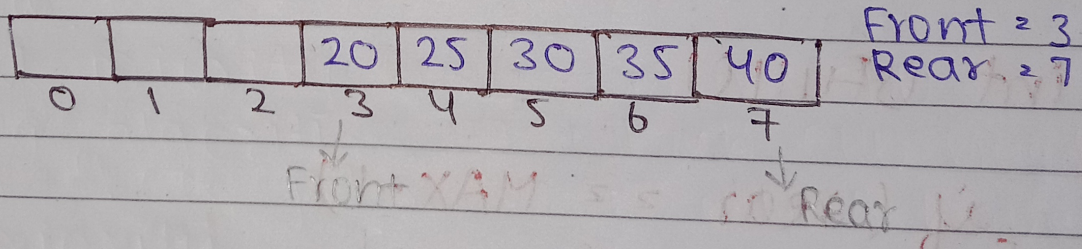
```
int peek()
{
  if (isEmpty())
  {
    printf("In Queue Underflow");
    exit(1);
  }
  return queue_arr[front];
}

int isEmpty()
{
  if(front == -1 || front == rear+1)
  {
    return 1;
  }
  else
  {
    return 0;
  }
}

int isfull()
{
  if(rear == MAX-1)
  {
    return 1;
  }
  else
  {
    return 0;
  }
}
```

```
void display()
{
   int i;
   if (isEmpty())
   {
      printf("\n Queue is empty");
      return;
   }
   printf("Queue is : ");
   for(i = front; i <= rear; i++)
   {
      printf("%d ", queue_arr[i]);
   }
   printf("\n\n");
}
```

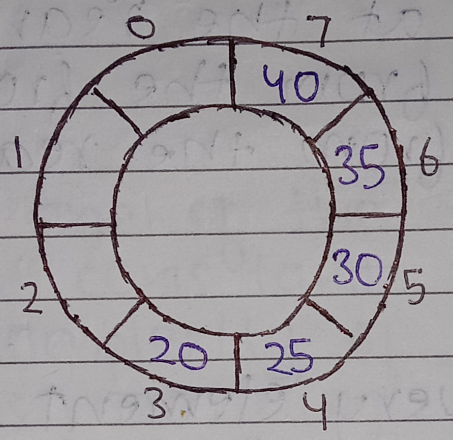
The drawbacks of this array implementation is that, consider the situation when rear is at the last position of array & front is not at the 0th position.



There are 3 spaces for adding the elements but we cannot insert any element in Queue because rear is at the last position of array.
The solution to avoid is we can shift all the elements of array to left, but this is practically not good because this will consume lot of time. Another efficient solution is Circular Queue.

Circular Queue :

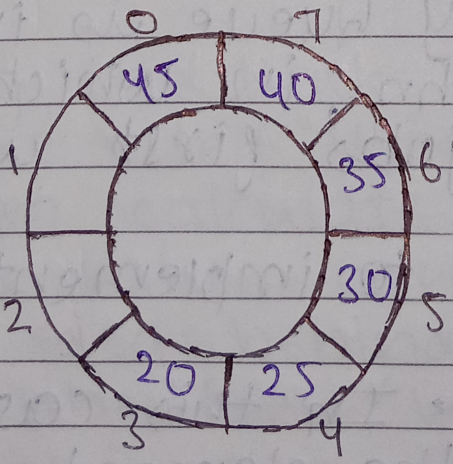
When Queue is implemented as an array, insertion is not possible after REAR reaches the position of array. There may be vacant position in the array but they can't be utilized. To overcome the limitation we use the concept of **Circular Queue**.



Front : 3
Rear : 7

Now, after the $(n-1)^{th}$ position occurs. If we want to insert an element, it can be inserted at 0^{th} position.

Insert 45,



DeQue :
It is a linear list in which elements can be inserted or deleted at either end of the list. The term deque is also called Double Ended Queue.
The four operations which performed in DeQue is :-
i) Insertion at the front end.
ii) Insertion at the rear end.
iii) Deletion from the front end.
iv) Deletion from the rear end.

Priority Queue :
In this Queue, every element of the Queue has some priority & it is processed based on its priority. High priority element is processed before the element which has less priority. If two element have same priority then FIFO rule will follow.
Example of priority Queue is in CPU scheduling algorithm, in which CPU needs to process those jobs first which have higher priority.
There are 2 ways to implement priority queue :-
i) Through Queue : In this case, insertion is simple because the element is simply added at the rear end. For performing deletion, first the element with highest priority is searched & then deleted.
ii) Through Sorted List : In this case insertion is costly because the element

is inserted at the proper place in the list based on its priority. Here, deletion is easy since the element with highest priority will always be in the beginning of the list.

—x——x——x——x——x——x——

Hashing

Hashing is one of the searching techniques that ufes a constant time.
The time complexity of Hashing is Big O(1)
The pervious searching techniques is :-
Linear Search & Binary Search.
In these searching techniques, the search -ing depends upon the number of eleme -nts.
The searching techniaue which provides a constant time is Hashing.

In Hashing, the Hash Table & Hash Func -tion are used. Using the Hash Functi -on, we can calculate the address at which the value can be stored.
The main idea behind the Hashing is to create the (Key / value) pairs.
If the key is given then the algorith -m computes the Index at which the value would be Stored.
we can write it as:-