Queue ADT

- A queue in C is basically **a linear data structure to store and manipulate the data elements**.

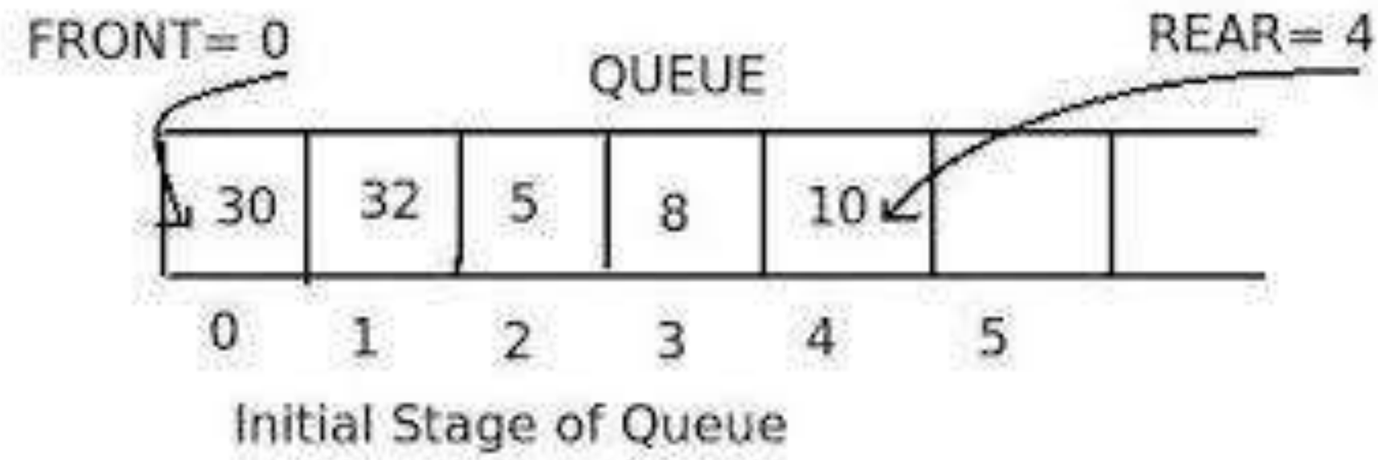
- It follows the order of First In First Out (FIFO).

- In queues, the first element entered into the array is the first element to be removed from the array.

- Queue is a linear data structure in which the **insertion and deletion** operations are performed at two different ends.

- In a queue data structure, adding and removing elements are performed at two different positions.

- The insertion is performed at one end and deletion is performed at another end.



Enqueue() : It inserts an element to the end of the queue by using Rear pointer.

Dequeue() : Removes the element from the front of the queue by using front pointer.

- **isFull() :** To check whether the queue is full or not.
- **isEmpty():** To check whether the queue is empty or not.

Ways of Implementation:

- **Array Implementation**
- **Linked List Implementation**

Array Implementation of Queue ADT

- An array is a linear data structure
- An array is a collection of variables in the same data type.
- we can’t group different data types in the array. Like, a combination of integer and char, char and float etc.
- Hence array is called as the homogeneous data type.
- Using **index** value, we can directly access the desired element in the array.

<u>**Operations of Queue ADT**</u>

- Implementation of queue using array starts with the creation of an array of **size n** and initialize two variables **FRONT** and **REAR** with **-1**which means currently **queue is empty**.
- The REAR value represents the index up to which value is stored in the queue and the FRONT value represents the index of the first element to be dequeued.

Enqueue

- Insert an element from the rear end into the queue.
- Element is inserted into the queue after checking the overflow condition n-1==REAR to check whether the queue is full or not.
- If **n-1==REAR** then this means the **queue is already full.**
- But if **REAR < n** means that **we can store an element in an array.**
- So increment the REAR value by 1 and then insert an element at the REAR index.

Routine for Enqueue Operation

```
void enQueue (int value)
  {
  if (rear == SIZE – 1)
  {
  printf (“Queue is Full”);
  }
  else
  {
  if ( front == -1)
  front = 0;
  }
  rear++;
  queue [rear] = value;
  }
```

Example:

int arr[7]={10,20,30,40,50,60,70};

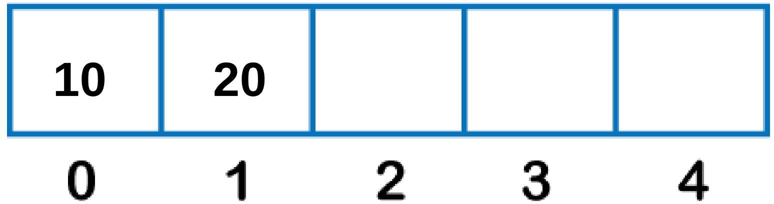
10	20	30	40	50	60	70	
0	1	2	3	4	5	6	7

0	1	2	3	4

Empty Queue

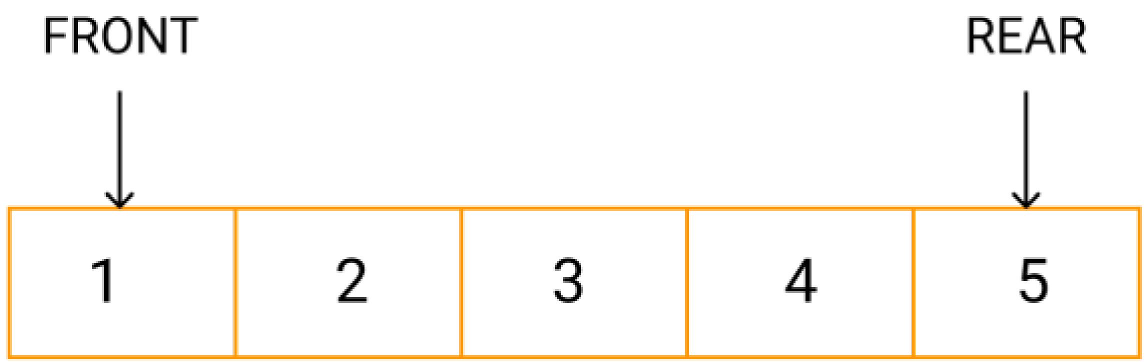
Front = -1

Rear = -1



Front = 0

Rear = 1



<u>Dequeue</u>

- Deleting an element from the FRONT end of the queue. Before deleting an element we need to check the underflow condition front == - 1 or front > rear to check whether there is at least one element available for the deletion or not.

- If **front == - 1** or **front > rear** then no element is available to delete.

- Else delete FRONT index element and Returns the FRONT value of queue.

- If **REAR==FRONT** then we **set -1** to both FRONT AND REAR

- Else we increment FRONT.

Routine for Dequeue Operation:

```
void deQueue( )
{
if ( front == -1 )
printf (“ Queue is Empty”);
else
{
printf (“ Deleted : %d”, queue [front]);
front ++;
if ( front > rear )    // only happens when the last element was dequeued
front = rear = -1;
}
}
```

Display

- It will traverse the queue and print all the elements of the queue.
- Check whether the queue is not empty or not.
- If empty, display that the queue is empty. we simply return from the function and not execute further inside the function.
- Else we will print all elements from FRONT to REAR by incrementing FRONT pointer.

Routine for Display Operation

```
void Display ( )
{
if ( front == -1)
printf (“ Queue is Empty”);
else
{
int i;
printf (“ Queue Elements are:”);
for ( i = front; i <= rear; i++)
printf (“%d:, queue[i]);
}
}
```

Linked List Implementation of Queue

- Implementing a queue using a linked list allows us to grow the queue as per the requirements, i.e., memory can be allocated dynamically.
- A queue implemented using a linked list will not change its behavior and will continue to work according to the FIFO principle.

Steps for implementing queue using linked list:

1. Enqueue Function

- Enqueue function adds an element to the end of the queue. The last element can be tracked using the rear pointer.
- First, build a new node with given data.
- Check if the queue is empty or not.
- If a queue is empty then, a new node is assigned to the front and rear.
- Else make next of rear as new node and rear as a new node.

2. Dequeue Function

- The dequeue function always removes the first element of the queue. For dequeue, the queue must contain at least one element, else underflow conditions will occur.
- Check if queue is empty or not.
- If the queue is empty, then dequeue is not possible.
- Else store front in temp
- And make next of front as the front.
- Delete temp, i.e., free (temp).

3. Print

- Print function is used to display the content of the queue. Since we need to iterate over each element of the queue to print it.
- Check if queue contains at least one element or not.
- If the queue is empty print “No elements in the queue.”
- Else, define a node pointer and initialize it with the front.
- Display data of node pointer until the next node pointer becomes NULL.

Node Structure

```
struct node
{
  int data;
  struct node * next;
};
```

Enqueue() operation on a queue

```
void Enqueue (int Element)
{
 struct node * newnode;
 newnode = (struct node *)malloc (sizeof (struct node));
 newnode - > data = Element;
 newnode - > next = NULL;
 if ((front == NULL) && (rear == NULL))
{
 front = rear = newnode;
 }
 else
 {
  rear - > next = newnode;
  rear = newnode;
 }
 }
```

Dequeue() operation on a queue

```
int dequeue()
{
if (front == NULL)
{
 printf("\nUnderflow\n");
 return -1;
}
 else
 {
  struct node * temp = front;
  int temp_data = front - > data;
  front = front - > next;
  free(temp);
  return temp_data;
 }
}
```

Display all elements of the queue

```
void display()
{
 struct node * temp;
 if ((front == NULL) && (rear == NULL))
 {
  printf("\nQueue is Empty\n");
  }
  else
  {
   temp = front;
   while (temp)
   {
     printf ("%d", temp - > data);
     temp = temp - > next;
    }
   }
  }
```