

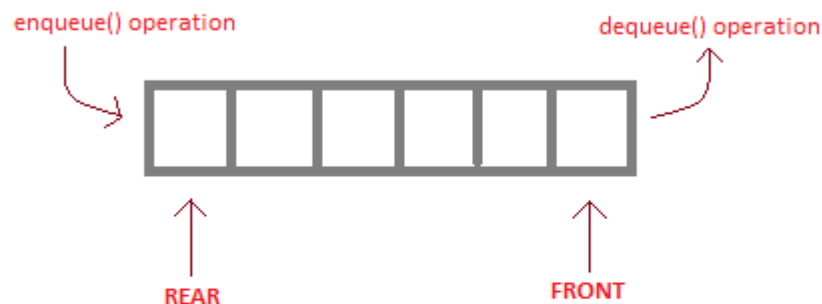
Queue

Queue is also an abstract data type or a linear data structure, just like [stack data structure](#), in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).

This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.

Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first.

The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



enqueue() is the operation for adding an element into Queue.

dequeue() is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Basic features of Queue

1. Like stack, queue is also an ordered list of elements of similar data types.
2. Queue is a FIFO(First in First Out) structure.
3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
4. `peek()` function is oftenly used to return the value of first element without dequeuing it.

Applications of Queue

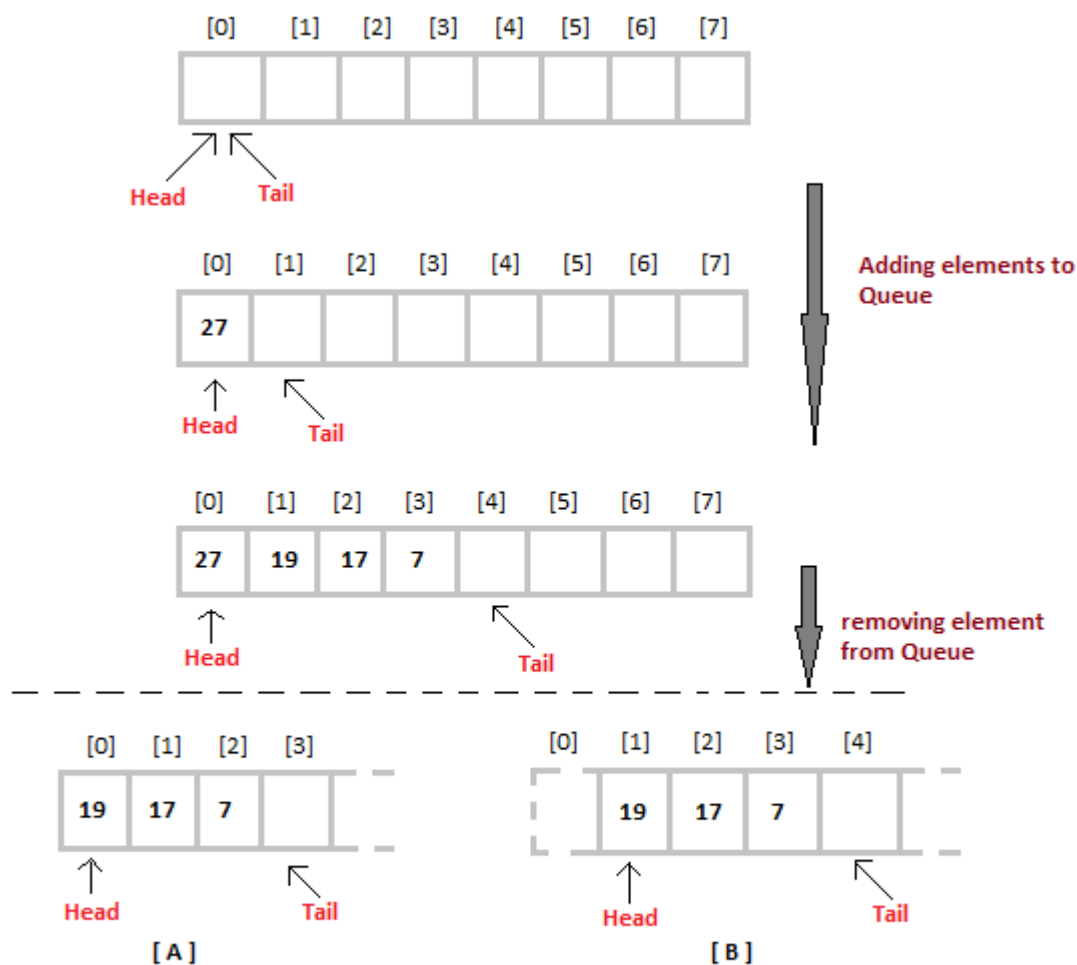
Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

Implementation of Queue Data Structure

Queue can be implemented using an [Array](#), [Stack](#) or [Linked List](#). The easiest way of implementing a queue is by using an Array.

Initially the **head**(FRONT) and the **tail**(REAR) of the queue points at the first index of the array (starting the index of array from 0). As we add elements to the queue, the **tail** keeps on moving ahead, always pointing to the position where the next element will be inserted, while the **head** remains at the first index.



When we remove an element from Queue, we can follow two possible approaches (mentioned [A] and [B] in above diagram). In [A] approach, we remove the element at **head** position, and then one by one shift all the other elements in forward position.

In approach [B] we remove the element from **head** position and then move **head** to the next position.

In approach [A] there is an **overhead of shifting the elements one position forward** every time we remove the first element.

In approach [B] there is no such overhead, but whenever we move head one position ahead, after removal of first element, the **size on Queue is reduced by one space** each time.

Algorithm for ENQUEUE operation

1. Check if the queue is full or not.
2. If the queue is full, then print overflow error and exit the program.
3. If the queue is not full, then increment the tail and add the element.

Algorithm for DEQUEUE operation

1. Check if the queue is empty or not.
2. If the queue is empty, then print underflow error and exit the program.
3. If the queue is not empty, then print the element at the head and increment the head.

```
#include <stdio.h>

#include<stdlib.h>

#define MAX 50

void insert();

void delete();

void display();

int queue_array[MAX];

int rear = - 1;

int front = - 1;

int main()

{

int choice;

while (1)

{

printf("1.Insert element to queue n");

printf("2.Delete element from queue n");

printf("3.Display all elements of queue n");

printf("4.Quit n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch(choice)

{

case 1:

insert();

break;

case 2:
```

```
delete();  
break;  
case 3:  
display();  
break;  
case 4:  
exit(1);  
default:  
printf("Wrong choice n");  
}  
}  
}
```

```
void insert()
{
int item;
if(rear == MAX - 1)
printf("Queue Overflow n");
else
{
if(front== - 1)
front = 0;
printf("Inset the element in queue : ");
scanf("%d", &item);
rear = rear + 1;
queue_array[rear] = item;
else
{
rear = rear + 1;
queue_array[rear] = item;
}
}
}
```

```
void delete()
{
if(front == - 1 || front > rear)
{
printf("Queue Underflow n");
return;
}
else
{
printf("Element deleted from queue is : %dn", queue_array[front]);
front = front + 1;
}
}
```



```
void display()
{
int i;
if(front == - 1)
printf("Queue is empty n");
else
{
printf("Queue is : n");
for(i = front; i <= rear; i++)
printf("%d ", queue_array[i]);
printf("n");
}
}
```

Implementation using linked list:

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node *ptr;
}*front, *rear, *temp, *front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Enqueue");
    printf("\n 2 - Dequeue");
    printf("\n 3 - Front element");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Display");
    printf("\n 7 - Queue size");
    create();
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                enq(no);
                break;
            case 2:
                deq();
                break;
            case 3:
```

```

        e = frontelement();
        if (e != 0)
            printf("Front element : %d", e);
        else
            printf("\n No front element in Queue as queue is empty");
        break;
    case 4:
        empty();
        break;
    case 5:
        exit(0);
    case 6:
        display();
        break;
    case 7:
        queuesize();
        break;
    default:
        printf("Wrong choice, Please enter correct choice ");
        break;
    }
}

/* Create an empty queue */
void create()
{
    front = rear = NULL;
}

/* Returns queue size */
void queuesize()
{
    printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
    else
    {
        temp=(struct node *)malloc(1*sizeof(struct node));
        rear->ptr = temp;
        temp->info = data;
        temp->ptr = NULL;

        rear = temp;
    }
    count++;
}

```

```

/* Displaying the queue elements */
void display()
{
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}

/* Dequeueing the queue */
void deq()
{
    front1 = front;

    if (front1 == NULL)
    {
        printf("\n Error: Trying to display elements from empty queue");
        return;
    }
    else
    {
        if (front1->ptr != NULL)
        {
            front1 = front1->ptr;
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = front1;
        }
        else
        {
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = NULL;
            rear = NULL;
        }
        count--;
    }
}

/* Returns the front element of queue */
int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}

/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear == NULL))
        printf("\n Queue empty");
}

```

```
    else  
        printf("Queue not empty");  
}
```