**Godavari College Of Engineering, Jalgaon.**

**Subject Name:** Data Structure **Teacher Name:** Prof.S.S.Shete

**Practical No**. : 4 **Date:**

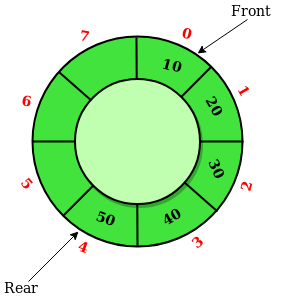
**Class: S**.E **Roll No:**

**Title:** Write a program to implement circular queue using arrays.

**Aim:** To implement program of circular queue using arrays.

**Theory:**

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called ‘Ring Buffer’. In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.



**Operations on Circular Queue:**

* Front: Get the front item from queue.
* Rear: Get the last item from queue.
* enQueue(value) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

Steps:

* 1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
  2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.
* DeQueue() This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

Steps:

* 1. Check whether queue is Empty means check (front==-1).
  2. If it is empty then display Queue is empty. If queue is not empty then step 3
  3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

**Program:-**

# include<stdio.h>

# define MAX 4

int cqueue\_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item)

{

if((front == 0 && rear == MAX-1) || (front == rear+1))

{

printf("Queue Overflow \n");

return;

}

if (front == -1) /\*If queue is empty \*/

{

front = 0;

rear = 0;

}

else

{

if(rear == MAX-1) /\*rear is at last position of queue \*/

rear = 0;

else

rear = rear+1;

}

cqueue\_arr[rear] = item ;

}

void del()

{

if (front == -1)

{

printf("Queue Underflow\n");

return ;

}

printf("Element deleted from queue is : %d\n",cqueue\_arr[front]);

if(front == rear) /\* queue has only one element \*/

{

front = -1;

rear=-1;

}

else

{

if(front == MAX-1)

front = 0;

else

front = front+1;

}

}

void display()

{

int front\_pos = front,rear\_pos = rear;

if(front == -1)

{

printf("Queue is empty\n");

return;

}

printf("Queue elements :\n");

if( front\_pos <= rear\_pos )

while(front\_pos <= rear\_pos)

{

printf("%d ",cqueue\_arr[front\_pos]);

front\_pos++;

}

else

{

while(front\_pos <= MAX-1)

{

printf("%d ",cqueue\_arr[front\_pos]);

front\_pos++;

}

front\_pos = 0;

while(front\_pos <= rear\_pos)

{

printf("%d ",cqueue\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

int main()

{

int choice,item;

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

rep:

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

printf("\n\t Input the element for insertion in queue : ");

scanf("%d", &item);

insert(item);

break;

case 2 :

del();

break;

case 3:

display();

break;

case 4:

printf("\n\n\t THanks For Vising Program");

break;

default:

printf("Wrong choice\n");

printf("\n\n==================================================\n");

}

if(choice!=4)

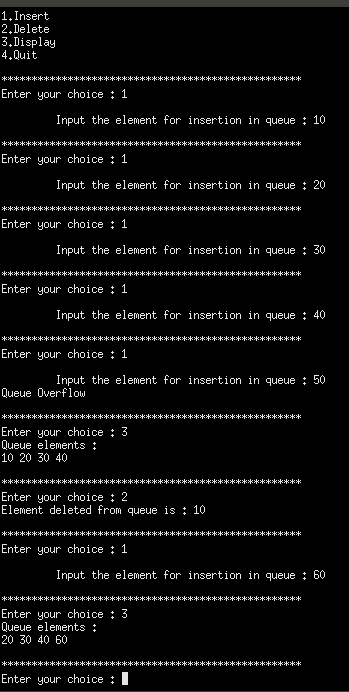
{

goto rep;

}

return 0;

}

**Output :-**

**Conclusion:-**