```c
// Circular Queue implementation in C

#include <stdio.h>

#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;

// Check if the queue is full
int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
  return 0;
}

// Check if the queue is empty
int isEmpty() {
  if (front == -1) return 1;
  return 0;
}

// Adding an element
void enQueue(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    printf("\n Inserted -> %d", element);
  }
}

// Removing an element
int deQueue() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty !! \n");
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
```

```
    // Q has only one element, so we reset the
    // queue after dequeing it. ?
    else {
      front = (front + 1) % SIZE;
    }
    printf("\n Deleted element -> %d \n", element);
    return (element);
  }
}

// Display the queue
void display() {
  int i;
  if (isEmpty())
    printf(" \n Empty Queue\n");
  else {
    printf("\n Front -> %d ", front);
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
    printf("\n Rear -> %d \n", rear);
  }
}

int main() {
  // Fails because front = -1
  deQueue();

  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);

  // Fails to enqueue because front == 0 && rear == SIZE - 1
  enQueue(6);

  display();
  deQueue();

  display();

  enQueue(7);
```

```
  display();

  // Fails to enqueue because front == rear + 1
  enQueue(8);

  return 0;
}
```

CIRCULAR QUEUE USING LIMKED LIST BY GOPI SIR IN C

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

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

struct node *f = NULL;
struct node *r = NULL;

void enqueue (int d)      //Insert elements in Queue
{
  struct node *n;
  n = (struct node *) malloc (sizeof (struct node));
  n->data = d;
  n->next = NULL;
  if ((r == NULL) && (f == NULL))
    {
      f = r = n;
      r->next = f;
    }
  else
    {
      r->next = n;
      r = n;
      n->next = f;
    }
}
```

```
void dequeue ()     // Delete an element from Queue
{
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else if (f == r)
    {
      f = r = NULL;
      free (t);
    }
  else
    {
      f = f->next;
      r->next = f;
      free (t);
    }


}

void display ()
{       // Print the elements of Queue
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else
    {
      do
  {
    printf (" %d", t->data);
    t = t->next;
  }
      while (t != f);
    }
}

int main ()
{
  enqueue (34);
  enqueue (22);
  enqueue (75);
  enqueue (99);
  enqueue (27);
```

```
  printf ("Circular Queue: ");
  display ();
  printf ("\n");

  dequeue ();
  printf ("Circular Queue After dequeue: ");
  display ();
  return 0;
}
```

CIRCULAR QUEUE IN C++ USING LINKED LIST

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct node{
    int data;
    struct node*next;
};
struct node*fr=0;
struct node*rear=0;
void enqueue(int x){
    struct node*newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=x;
    newnode->next=0;
    if(rear==0){
        fr=rear=newnode;
        rear->next=fr;
    }else{
        rear->next=newnode;
        rear=newnode;
        rear->next=fr;
    }
}
void display(){
    struct node*temp;
    if(fr==0&&rear==0) cout<<"Queue is empty"<<endl;
    else{
        cout<<"The elements of queue are : ";
        while(temp->next!=fr){
            cout<<temp->data<<" ";
            temp=temp->next;
        }cout<<temp->data<<" "<<endl;
```

```
    }
}
void dequeue(){
    struct node *temp;
    temp=fr;
    if(fr==0&&rear==0) cout<<"Queue is empty"<<endl;
    else if(fr==rear){
        fr=rear=0;
        free(temp);
    }else{
        fr=fr->next;
        rear->next=fr;
        free(temp);
    }
}
void peek(){
    struct node *temp;
    if(temp==0&&rear==0) cout<<"Queue is empty"<<endl;
    else cout<<"The front element was "<<fr->data<<endl;
}
int main(){
    enqueue(2);
    enqueue(-1);
    enqueue(5);
    display();
    dequeue();
    peek();
    return 0;
}
```

QUEUE USING ARRAYS

```
#include<iostream>
#define N 5
using namespace std;
int q[N];
int fr=-1,rear=-1;
void enqueue(int x){
    if(rear==N-1) cout<<"Overflow"<<endl;
    else if(fr==-1&&rear==-1){
        fr=rear=0;
        q[rear]=x;
    }else{
        rear++;
        q[rear]=x;
```

```
    }
}
void dequeue(){
    if(fr==-1&&rear==-1) cout<<"Underflow"<<endl;
    else if(fr==rear) fr=rear=-1;
    else{
        cout<<"The deleted elemnet in queue was "<<q[fr]<<endl;
        fr++;
    }
}
void display(){
    if(fr==-1&&rear==-1) cout<<"The queue is empty"<<endl;
    else for(int i=fr;i<rear+1;i++) cout<<q[i]<<endl;;
}
void peek(){
     if(fr==-1&&rear==-1) cout<<"The queue is empty"<<endl;
     else cout<<"the front element is "<<q[fr]<<endl;
}
int main(){
    enqueue(2);
    enqueue(5);
    enqueue(-1);
    display();
    peek();
    dequeue();
    peek();
    display();
    return 0;
}
```