**1.a) Implementation of Queue**

#include<stdio.h>

#define MAX\_SIZE 10

int queue[MAX\_SIZE],j;

int front=-1,rear=-1;

int enqueue (int x) {

if(rear==MAX\_SIZE-1){

printf(“over flow”);

return ;

}

if(front==-1){

front=0;

}

rear++;

queue[rear]=x;

printf(“enqueued %d to the queue\n”,x);

return rear;

}

int dequeue(){

if(front==-1||front>rear){

printf(“deletion not possible\n”);

front=rear=-1;

return -1;

}

else{

printf(“dequeued %d from the queue\n”,queue[front]);

front++;

}

return(front);

}

int main(){

enqueue(40);

enqueue (38);

enqueue(20);

enqueue(39);

dequeue();

dequeue();

dequeue();

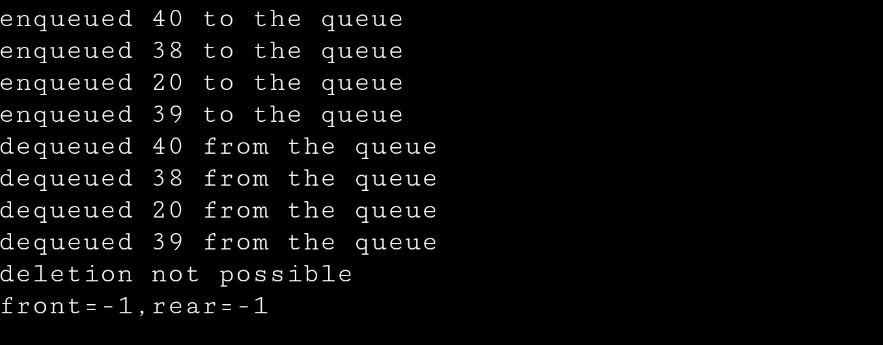
dequeue();

dequeue();

printf(“front=%d,rear=%d”,front,rear);

return 0;

}



**1.b)Circular Queue**

#include <stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

int isFull() {

If ((front == rear + 1) || (front == 0 && rear == SIZE – 1)) return 1;

return 0;

}

int isEmpty() {

if (front == -1) return 1;

return 0;

}

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);

}

}

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;

}

else {

front = (front + 1) % SIZE;

}

printf(“\n Deleted element -> %d \n”, element);

return (element);

}

}

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() {

enQueue(1);

enQueue(8);

enQueue(0);

enQueue(7);

enQueue(4);

enQueue(5);

display();

deQueue();

display();

enQueue(7);

deQueue();

enQueue(6);

return 0;

