EX.NO:5B) ARRAY IMPLEMENTATION OF QUEUE

AIM:

To write a C Program using array based implementation of queue.

ALGORITHM:

Step 1: Initialize the queue variables front =0 and rear = -1
Step 2: Read the queue operation type.
Step 3: Check the queue operations status.
i). If it is Insertion then do the following steps
1. Check rear < queue_size is true increment the rear by one and read the queue element and also display queue. otherwise display the queue is full.
2. Go to step2.
ii). If it is deletion then do the following steps
1. Check rear< front is true then display the queue is empty.
2. Move the elements to one step forward (i.e. move to previous index).
3. Decreases the rear value by one (rear=rear-1).
4. Display queue
5. Go to step2.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
// A structure to represent a queue
struct Queue
{
    int front, rear, size;
    unsigned capacity;
    int* array;
};
 // function to create a queue of given capacity.
// It initializes size of queue as 0
struct Queue* createQueue(unsigned capacity)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;  // This is important, see the enqueue
    queue->array = (int*) malloc(queue->capacity * sizeof(int));
    return queue;
}
 // Queue is full when size becomes equal to the capacity
int isFull(struct Queue* queue)
{  return (queue->size == queue->capacity);  }
 // Queue is empty when size is 0
int isEmpty(struct Queue* queue)
{  return (queue->size == 0); }

void enqueue(struct Queue* queue, int item)
{
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1)%queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    printf("%d enqueued to queue\n", item);
}
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1)%queue->capacity;
    queue->size = queue->size - 1;
    return item;
}
int front(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->front];
}
```

```
int rear(struct Queue* queue)
{
   if (isEmpty(queue))
      return INT_MIN;
   return queue->array[queue->rear];
}
int main()
{
   struct Queue* queue = createQueue(1000);
   enqueue(queue, 10);
   enqueue(queue, 20);
   enqueue(queue, 30);
   enqueue(queue, 40);
   printf("%d dequeued from queue\n", dequeue(queue));
   printf("Front item is %d\n", front(queue));
   printf("Rear item is %d\n", rear(queue));
   return 0;
}
```

OUTPUT:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```

RESULT:

Thus the C program for array based implementation of Queue has been executed and verified successfully

6. LINKED LIST IMPLEMENTATION OF STACK AND QUEUE