Name : Kshitij Vyas
SE 3 C
Roll No. : 62

Experiment no 5

Aim: To implement Circular Queue ADT using array

Objective:

Circular Queue offer a quick and clean way to store FIFIO data with maximum size

Algorithm

1. INIT(QUEUE,FRONT,REAR,COUNT)
2. INSERT-ITEM(QUEUE, FRONT, REAR, MAX, COUNT, ITEM)
3. REMOVE-ITEM(QUEUE, FRONT, REAR, COUNT, ITEM)
4. FULL-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,FULL)
5. EMPTY-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,EMPTY)

INIT(QUEUE,FRONT,REAR,COUNT)

This algorithm is used to iniϴ alize circular queue.

1. FRONT := 1;

2. REAR := 0;

3. COUNT := 0;4. Return;

INSERT-ITEM(QUEUE, FRONT, REAR, MAX, COUNT, ITEM)

This algorithm is used to insert or add item

into a circular queue.

1.

If (COUNT = MAX) then

a. Display “Queue overflow”;

b. Return;

2.

Otherwise

a. If (REAR = MAX) then

i.REAR := 1;

b. Otherwise

i. REAR := REAR + 1;

c. QUEUE(REAR) := ITEM;

d. COUNT := COUNT + 1;

3.

Return;

REMOVE-ITEM(QUEUE, FRONT, REAR, COUNT, ITEM)

This algorithm is used to remove or delete item

from the circular queue.

1.

If (COUNT = 0) then

a. Display “Queue underflow”;

b. Return;

2.

Otherwise

a. ITEM := QUEUE(FRONT)l

b. If (FRONT =MAX) then

i.FRONT := 1;

c. Otherwise

i.FRONT := FRONT + 1;

d. COUNT := COUNT + 1;3.

Return;

EMPTY-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,EMPTY)

This is used to check if the queue is empty or not.

1.

If(COUNT = 0) then

a. EMPTY := true;

2.

Otherwise

a. EMPTY := false;

3.

Return ;

FULL-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,FULL)

This algorithm is used to check if the queue is full or not.

1.

If (COUNT = MAX) then

a. FULL := true;

2.

Otherwise

a. FULL := false;

3.

Return ;

Circular Queue implementation in C

```
#include <stdio.h>

#define SIZE 5

int items[SIZE];
```

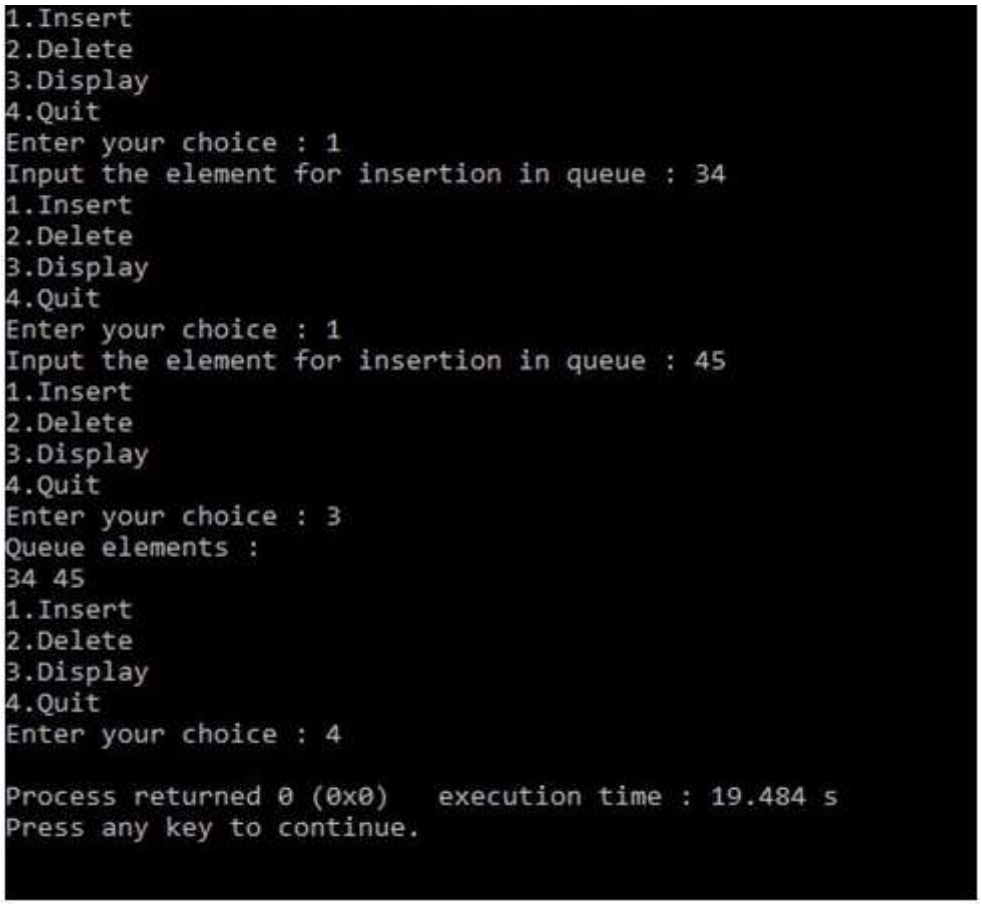
```
int front = -1, rear = -1;

// 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;
}
```

Output:



Conclusion:

The Circular Queue is similar to a Linear Queue in the sense that it follows the FIFO (First In First Out) principle but differs in the fact that the last posiϴon is connected to the first positon, replicating a circle.