Experiment No. : 4

Title: Implementation of Static Circular Queue

Batch: B-1 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.: 4

Aim: Write a menu driven program to implement a static circular queue using counter method supporting following operations.

1. Create empty queue,
2. Insert an element on the queue,
3. Delete an element from the queue,
4. Display front, rear element or
5. Display all elements of the queue.

Resources Used: Turbo C/ C++ editor and compiler.

Theory:

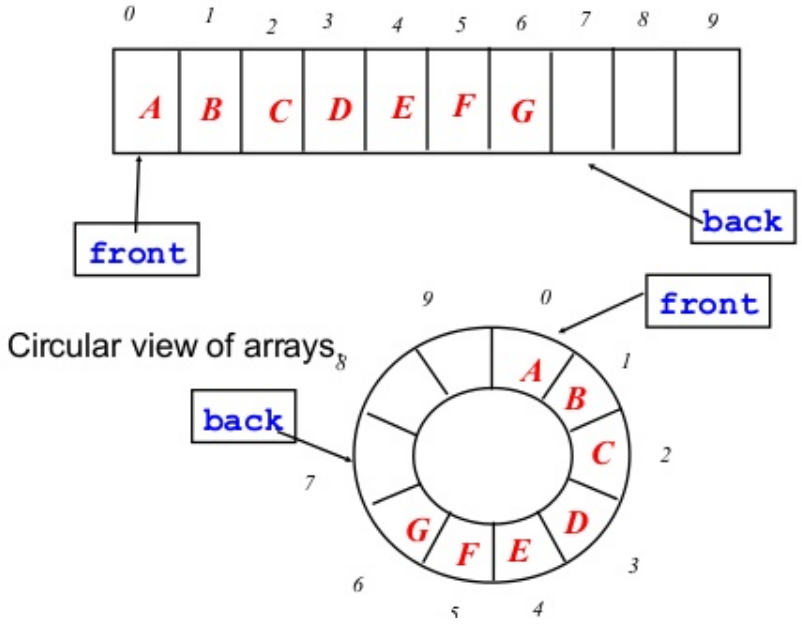
Queue -

A queue is an ordered list in which insertion and deletion happens two different ends. The insertion happens from the *rear* and the deletion takes place at the *front*. It works with the FIFO concept i.e. first in first out.

Basic operations of Queue are enqueue, dequeue, isempty, etc.

- **Circular Queue** : It is a variation of queue in which start of the queue is immediately next of end of the queue.

- To implement queue, it is best to view arrays as circular structure



Methods : There are two standard methods to implement a circular queue.

- **Method 1 – Counter method** – in this we keep a counter to track underflow and overflow of the queue. So one can initialize queue front = 0, rear = 0, and count=0.
- **Method 2 – One empty slot between front and rear of the queue** – in this we keep front and rear apart by one empty slot, which helps in differentiating between underflow and overflow on circular queue.

Algorithm :

1. **createQueue():**

This method creates an empty queue by setting initial values of front, rear and count to zero.

2. **void enqueue(typedef val) :**

This operation adds an item at the rear of the queue. Before insert operation, ensure that there is a room for the new item. If there is not enough room, then the queue is in an 'Overflow' state.

3. **typedef dequeue() :**

This operation removes the item at the *front* of the *Queue* and returns it to the user. Because the front item has been removed, the next item becomes the front. When the last item in the queue is deleted, it must be set to its empty state. If *dequeue* is called when the queue is empty, then it's in an 'Underflow' state.

4. **typedef getFront() :**

This operation will return the item present at *front* position at that instance.

5. **typedef getRear() :**

This operation will return the item present at *rear* position at that instance.

6. **void displayAll():**

This operation will display all item present in the queue at that instance.

7. **boolean isEmpty() :**

This operation will check whether *Queue* is empty or not at a given instance. The function will return

0 if *Queue* is not empty.
1 if *Queue* is empty.

6. boolean isFull() :

This operation will check whether *Queue* is full or not at a given instance. The function will return

0 if *Queue* is not full.
1 if *Queue* is full.

7. int size() :

This operation will count the total number of elements present in the *Queue* at a given instance and return the count.

0 if Queue is empty.
n if Queue is having n no. of elements.

Activity: students are expected to implement a circular queue using **method 1.** Make use of circular increment(e.g. front=(front+1)%maxsize)

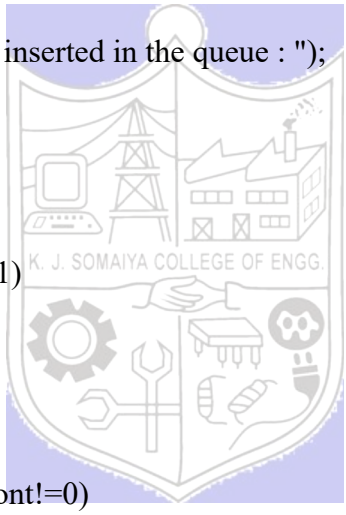
NOTE : All functions should be able to handle boundary(exceptional) conditions.

Results: A program implementing a counter based solution depicting the correct behavior of circular queue and capable of handling all possible exceptional conditions and the same is reflected clearly in the output.

Outcome:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
struct queue{
   int front, rear, count, b[MAX];
   }q;
int i,val;
void main()
{
   int choice;
    printf("\n1. Create empty queue\n2. Insert an element in the queue\n3. Delete an element from the queue\n4. Display front and rear elements of the queue\n5. Display all elements of the queue\n6. Exit");
   printf("\nChoose an option: ");
   scanf("%d",&choice);
   switch(choice)
   {
   case 1:
      q.front=0;
      q.rear=-1;
      q.count=0;
      main();
      break;
   case 2:
      enqueue();
      main();
      break;
   case 3:
      dequeue();
      printf("\nThe deleted element is: %d",val);
      main();
      break;
   case 4:
      getFront();
      printf("\nThe front element is: %d",q.b[q.front]);
```

```
        getRear();
        printf("\nThe rear element is: %d",q.b[q.rear]);
        main();
        break;
    case 5:
        displayAll();
        main();
        break;
    case 6:
        break;
    default:
        printf("Choose a valid option!");
        main();
        break;
    }
}
void enqueue()
{
    int num;
    printf("\nEnter the number to be inserted in the queue : ");
    scanf("%d", &num);
    if(q.count==MAX)
    {
        printf("\nOVERFLOW!");
    }
    else if(q.front==0 && q.rear==-1)
    {
        q.rear=(q.rear + 1)%MAX;
        q.b[q.rear]=num;
        q.count++;
    }
    else if(q.rear==MAX-1 && q.front!=0)
    {
        q.rear=(q.rear + 1)%MAX;
        q.b[q.rear]=num;
        q.count++;
    }
    else
    {
        q.rear++;
        q.b[q.rear]=num;
        q.count++;
    }
}
int dequeue()
{
    if(q.count==0)
        {
```



```
        printf("\nUNDERFLOW");
        return -1;
    }
    val = q.b[q.front];
    if(q.front==q.rear)
        q.front=q.rear=-1;
    else
        {
            if(q.front==MAX-1)
                q.front=0;
            else
                q.front=(q.front + 1)%MAX;
        }
    return val;
}
int getFront()
{
  if(q.count==0)
    {
        printf("\nQUEUE IS EMPTY!");
        return -1;
    }
  else
    {
        return q.b[q.front];
    }
}
int getRear()
{
  if(q.count==0)
    {
        printf("\nQUEUE IS EMPTY!");
        return -1;
    }
  else
    {
        return q.b[q.rear];
    }
}
void displayAll()
{
  if (q.count == 0)
    {
    printf("\nQUEUE IS EMPTY!");
    }
  else
    {
    if (q.front <= q.rear)
```

```
    {
       for (i = q.front; i <= q.rear; i++)
       {
          printf("\t%d", q.b[i]);
       }
    }
    else
       {
       for (i = q.front; i < MAX; i++)
       {
          printf("\t%d", q.b[i]);
       }
       for (i = 0; i != q.rear; i = (i + 1) % MAX)
       {
          printf("\t%d", q.b[i]);
       }
       }
    }
}
```

```
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 1

1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 2

Enter the number to be inserted in the queue : 1

1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 2

Enter the number to be inserted in the queue : 7
```

```
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 2

Enter the number to be inserted in the queue : 3

1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 2

Enter the number to be inserted in the queue : 8

1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 2

Enter the number to be inserted in the queue : 4
```

```
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 4

The front element is: 1
The rear element is: 4
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 5
        1       7       3       8       4
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 3

The deleted element is: 1
```

```
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 5
        7       3       8       4
1. Create empty queue
2. Insert an element in the queue
3. Delete an element from the queue
4. Display front and rear elements of the queue
5. Display all elements of the queue
6. Exit
Choose an option: 6

Process returned 4200086 (0x401696)   execution time : 170.472 s
Press any key to continue.
```

Conclusion:

The experiment successfully implemented a menu-driven program for a static circular queue using the counter method. The program demonstrates the correct functionality of the specified operations.

Grade: AA / AB / BB / BC / CC / CD /DD:

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.