REPUBLIQUE DU CAMEROON
PAIX-Travail-Patrie
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR

FACULTE D'INGINERIE ET TECHGNOLOGIE

REPUBLIC OF CAMEROON
Peace-Work-Fatherland
MINISTER OF HIGHER EDUCATION

FACULTY OF ENGINEERING AND TECHNOLOGY

********** UNIVERSITY OF BUEA **********

**********FACULTY OF ENGINEERING AND TECHNOLOGY**********

**********DEPARTMENT OF COMPUTER ENGINEERING**********

COURSE TITLE: ALGORITMS AND DATA STRUCTURES

COURSE CODE: CEF 341

IMPLEMENTATION OF QUEUES USING CIRCULAR ARRAYS

NAME: KINGO KINGSLEY KAAH

MATRICULE: FE22A233

IMPLEMENTATION OF QUEUES USING CIRCULAR ARRAYS

A circular queue is similar to a simple queue in that, they both are based on the FIFO principle, except for the fact that the last position is connected to the firs position in a circular queue to form a circle.

Here, we will make use of the rear and the front. Now, since at some point, the rear becomes the front and front becomes the rear, upon creating the queue, the rear will be initialized to -1 and the front to 0 as shown below

```c
#include <stdio.h>
#define maxSize 1000

int queue[maxSize];
int front=0;
int rear=-1;
int n;
int counter = 0;

void create(){
      front = 0;
      rear = -1;
      printf("WELCOME! YOUR QUEUE HAS BEEN CREATED SUCCESSFULLY.....  \n");
}
```

Also, we have a maxSize which represents the maximum queue size and n which will be inputed by the user at runtime. Also, we have the counter which has been initialized to 0. It will behave like the front in the case of queue implementation using a simple array, since the front in this case is changing.

I. Function to Enqueue

This function enables users insert in a particular order, elements into a queue. In other to enqueue, one has to first check the queue to know whether or not it is full and in this case, the queue is full when counter = n, where n is the size to be defined by the user at runtime. If this condition is true, a message is displayed telling the user that the queue is full. After which it checks if rear =n -1, and then resets it to -1, if true

```c
void enqueue(int x) {
    if (counter == n) {
        printf("QUEUE IS FULL CANT'T ENQUEUE!\n");
        return;
    }
    if (rear == n-1) {
        rear = -1;
    }

    queue[++rear] = x;
    counter++;
    printf("INSERTED -> %d\n", x);
}
```

The value of the rear(end) is incremented by one and at that index, the element is inserted(enqueued)and the counter increments too The inserted value is printed to the user.

II. Function to Dequeue

For our program to be able to meet the basic operations performed on a queue data structure, the dequeue function cannot be left out. To dequeue, we first of all check to see if the queue is empty. To do so, we check to see if the head, counter = 0; and a message is printed telling the user that the queue is empty and hence cannot dequeue.

```c
int dequeue() {
    if (counter == 0) {
        printf("QUEUE IS EMPTY, CAN'T DEQUEUE!\n");
        return -1;
    }
    int x = queue[front++];
    if (front == n) {
        front = 0;
    }
    counter--;
    printf("DELETED -> %d\n", x);
    return x;
}
```

The element to be dequeued x is assigned the queue element at index front+1. Again front is checked to see whether it is equal to the queue size, n and if true, it is reset to 0. Here, the counter decrements and the dequeued element, x is printed to the user.

III. Function to get the queue size

This function basically gives the number of elements found in the queue at any instance in the code.
In other to determine the size of the queue, we check whether the queue is empty, that is by checking whether the counter = 0
If that's the case, the queue size remains zero. If rear(end) is different, we use a for loop and another variable, l initialized to zero which is incremented as i increments from head to end; for(int i = front; i<=rear;i++){ l++; }

```c
void SizeOfQueue(){
    int l = 0;
    if(counter == 0){
        printf("QUEUE SIZE IS:  0\n");
        return;
    }
    for(int i = front; i<=rear;i++){
        l++;
    }
    printf("THE QUEUE SIZE IS: %d",l);
    printf("\n");
}
```

The value of l is stored and hence printed as the queue size.

IV. Function to display Queue elements

In this function, we check if the queue is empty, since we can't display what is not there. Provided the queue is not empty, a while loop is used to achieve the purpose of the function. In this case, the loop has 1 as its condition such that it is always true.

```
56  void display() {
57      if (counter== 0) {
58          printf("QUEUE IS EMPTY, NOTHING TO DISPLAY!\n");
59          return;
60      }
61      int i = front;
62      printf("QUEUE ELEMENTS ARE: ");
63      while (1) {
64          printf("%d ", queue[i]);
65          if (i == rear) {
66              break;
67          }
68          i = (i + 1) % n;
69      }
70      printf("\n");
71  }
```

i = front and while 1, the elements at index i(index front are being printed). It checks to see if i = rear or end and breaks when condition is true. After which i=(i+1)%n. in this logic, we avoid the case where the queue will return nothing to display whereas there are still elements in the queue.

V. Functions for the head and end of the Queue

These two functions basically prints the index of the head(front) and the tail(rear), as well as the elements at these indexes.

```
72  void headOfQueue(){
73      printf("\nFRONT--> %d\n",front);
74      printf("ELEMENT AT INDEX %d IS: %d\n",front,queue[front]);
75  }
76  void endOfQueue(){
77      printf("\nREAR--> %d\n",rear);
78      printf("ELEMENT AT INDEX %d IS: %d\n",rear,queue[rear]);
79  }
```

The above code will display when called in the main function, the head and tail of the queue and the elements at their various indexes.

The main program

The main function is responsible for calling all other functions in the entire program for execution. In this function, all other functions in the program were called. In order to make the program flexible and user friendly:

- The size of the queue is inputed at runtime and in this case, we have n which must be within the range of 1 and maxSize
- The user decides when to carryout any operation and when to quit executing that particular operation.
- A switch case was used to enable the user make a choice. That is; when to enque, dequeue, display and more. Also, a loop was added in the case of the enqueue that enables the user to keep enqueueing until a certain value(-1) in this case is inputed. Furthermore, a loop was also added in the dequeue case which asks the user how many elements the user would like to dequeue.

All of these helped to achieve the main purpose of a queue as well as other functionalities. Below is the code for the main function.

```
81  int main() {
82      int x, choice, times;
83      printf("ENTER THE DESIRED SIZE OF THE QUEUE..... ");
84      scanf("%d",&n);
85      if(n<=0||n>maxSize){
86          printf("ENTER A SIZE BETWEEN 1 AND %d\n",maxSize);
87          return 0;
88      }
89      create();
90      printf("ENTER THE OPERATION OF THE QUEUE OPERATION YOU WANT TO PERFORM\n");
91      printf(".......********************........\n");
92      while(1){
93          printf("1. ENQUEUE\n");
94          printf("2. DEQUEUE\n");
95          printf("3. DISPLAY\n");
96          printf("4. QUEUE SIZE\n");
97          printf("5. HEAD OF QUEUE!\n");
98          printf("6. END OF QUEUE!\n");
99          printf("7. QUIT EXECUTION!\n");
100         printf("\nENTER YOUR CHOICE...  ");
101         scanf("%d",& choice);
102         switch (choice){
103             case 1: printf("ENTER THE VALUE YOU WANT TO ENQUEUE\n");
104                 printf("ENTER -1 TO QUIT ENQUEUE\n");
105                 for(int i=0;i<=n;i++){
106                     scanf("%d",&x);
107                     if(i==n){
108                         printf("QUEUE IS FULL!\n");
109                         continue;
110                     }
111                     if(x==-1){
112                         break;
113                     }
```

```
                            enqueue(x);
                            display();
                        }
                break;

                case 2: printf("ENTER NUMBER OF ELTS TO DEQUEUE\n");
                         scanf("%d",&times);
                         for(int i=0;i<times;i++){
                             dequeue();
                         }
                         display();
                         SizeOfQueue();
                break;

                case 3: display();
                break;

                case 4: SizeOfQueue();
                break;
                case 5: headOfQueue();
                break;
                case 6: endOfQueue();
                break;
                case 7: printf("EXECUTION ENDED. THANKS\n");
                        return 0;
                break;

                default: printf("ENTER A VALID CHOICE (1/2/3/4/5)\n");
            }
        }
        return 0;
}
```

The output of the above code is as shown below

```
ENTER THE DESIRED SIZE OF THE QUEUE........ 4
WELCOME! YOUR QUEUE HAS BEEN CREATED SUCCESSFULLY.....
ENTER THE OPERATION OF THE QUEUE OPERATION YOU WANT TO PERFORM
.......********************........
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. QUEUE SIZE
5. HEAD OF QUEUE!
6. END OF QUEUE!
7. QUIT EXECUTION!

ENTER YOUR CHOICE...  1
ENTER THE VALUE YOU WANT TO ENQUEUE
ENTER -1 TO QUIT ENQUEUE
3
INSERTED -> 3
QUEUE ELEMENTS ARE: 3
5
INSERTED -> 5
QUEUE ELEMENTS ARE: 3 5
6
INSERTED -> 6
QUEUE ELEMENTS ARE: 3 5 6
8
INSERTED -> 8
QUEUE ELEMENTS ARE: 3 5 6 8
9
QUEUE IS FULL!
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. QUEUE SIZE
5. HEAD OF QUEUE!
6. END OF QUEUE!
7. QUIT EXECUTION!

ENTER YOUR CHOICE...
```

```
ENTER YOUR CHOICE...  2
ENTER NUMBER OF ELTS TO DEQUEUE
2
DELETED -> 3
DELETED -> 5
QUEUE ELEMENTS ARE: 6 8
THE QUEUE SIZE IS: 2
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. QUEUE SIZE
5. HEAD OF QUEUE!
6. END OF QUEUE!
7. QUIT EXECUTION!

ENTER YOUR CHOICE...  5

FRONT--> 2
ELEMENT AT INDEX 2 IS: 6
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. QUEUE SIZE
5. HEAD OF QUEUE!
6. END OF QUEUE!
7. QUIT EXECUTION!

ENTER YOUR CHOICE...  6

REAR--> 3
ELEMENT AT INDEX 3 IS: 8
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. QUEUE SIZE
5. HEAD OF QUEUE!
6. END OF QUEUE!
7. QUIT EXECUTION!

ENTER YOUR CHOICE...
```

Following the above codes, desired results have been obtained from the execution of the codes as shown above.