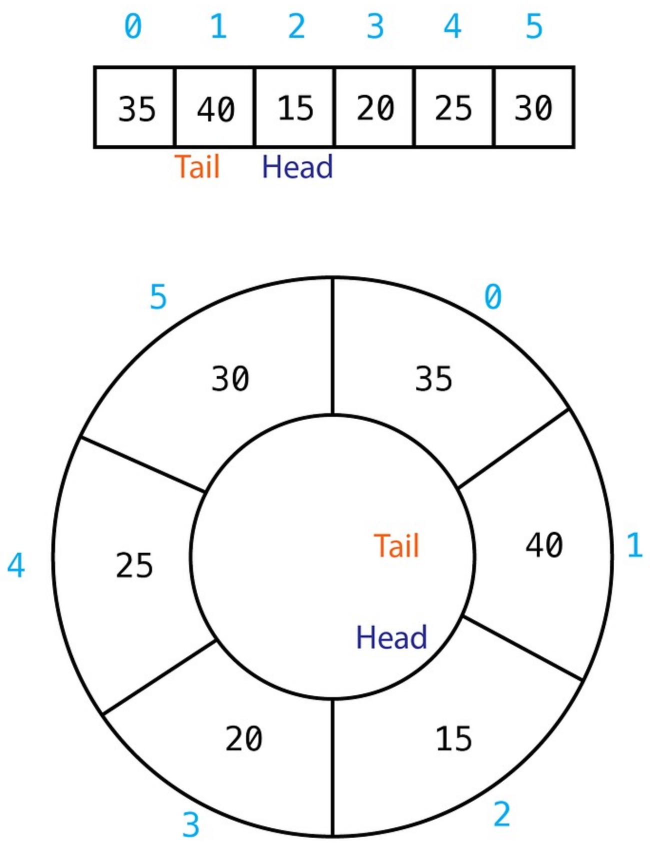
LAB 1

CIRCULAR QUEUE IMPLEMENTATION IN C

A circular queue is a data structure that follows the First In, First Out (FIFO) principle. It shares similarities with a regular queue but with a circular arrangement of elements. In a circular queue, when the last position is occupied and there is a new element to be enqueued, the next position is the first one. This creates a circular pattern, hence the name.

Key Definitions:

Circular Queue Implementation: In C, a circular queue can be implemented using an array. The array stores the elements, and two variables (front and rear) keep track of the indices representing the front



```
Q = circularQueue(6)
Q.Enqueue(5)
Q.Enqueue(10)
Q.Enqueue(15)
Q.Enqueue(20)
Q.Enqueue(25)
Q.Enqueue(30)
Q.Dequeue()
Q.Dequeue()
Q.Enqueue(35)
Q.Enqueue(40)
```

and rear of the queue. The circular arrangement is achieved by considering the next position modulo the maximum size of the queue.

Circular Queue: The main data structure that holds elements in a First In, First Out (FIFO) order with a circular arrangement.

Front: The index pointing to the front element of the circular queue.

Rear: The index pointing to the rear element of the circular queue.

Enqueue: The operation of adding an element to the rear of the circular queue.

Dequeue: The operation of removing the element from the front of the circular queue.

Enqueue:

Description: Adds an element to the rear of the circular queue.

Algorithm:
-Check if the queue is full.
-If not, increment the rear index and store the new element at that index.

Dequeue:

Description: Removes the element from the front of the circular queue.

Algorithm:
-Check if the queue is empty.
-If not, retrieve the element at the front index and increment the front index.

Display Front:

Description: Retrieves the element at the front without removing it.

Algorithm:
-Check if the queue is empty.
-If not, return the element at the front index.

DISCUSSION

The C program presented here is a direct implementation of a circular queue with fundamental operations like enqueue, dequeue, display of all elements, display of the front element, and appropriate handling of overflow and underflow conditions. Similar to the stack implementation, this program also employs a menu-driven interface for user interaction. It ensures clear communication with the user in case of queue overflow or underflow. The code features basic input handling using getchar() and scanf(), maintaining a user-friendly design. As with the stack implementation, further refinement could involve incorporating error handling for invalid user inputs and improving the display format for better readability. Overall, this implementation provides a solid foundation for understanding and building upon circular queue-related concepts in C programming.

CONCLUSION

This C program offers a functional implementation of a circular queue, allowing users to perform basic queue operations. It effectively checks for queue overflow and underflow conditions, providing informative messages to the user. The menu-driven approach enhances user interaction, allowing individuals to work with the circular queue seamlessly.

CODE

```
#include<stdio.h>
#include<stdlib.h>

#define maxsize  4
 int queue[maxsize], rear = 0, front = 0;

void enqueue(int item){
    queue[rear] = item;
    rear=(rear+1)%maxsize;
    printf(">>>Item added!\n");
}
int dequeue(){
    int element;
    element = queue[front];
    front=(front+1)%maxsize;
    printf("item dequeued!\n");
    return element;
}

int main(){
              int c, item, i;
          printf("Welcome to the Queue implementation in c\n");
    printf("Type (1.Enqueue  2.Dequeue  3.Display front element  4.Display all 
element  5.exit): \n");
    scanf("%d", &c);

    do {
        switch (c) {
        case 1:
            if ((rear+1)%maxsize==front) {
                printf(">>>Queue is full!\n");
            } else {
                printf("Type integer to add to the queue : ");
                scanf("%d", &item);
                enqueue( item);
            }
            break;

        case 2:
            if (front == rear) {
                printf(">>>Queue is empty!\n");
            } else {
                dequeue();
            }
            break;
        case 3:
            printf(">>>The front element is %d\n", queue[front]);
            break;

        case 4:
            for (i = front; i < rear; i++) {
                printf("%d ", queue[i]);
            }
            printf("\n");
            break;

        case 5:
            exit(0);
            break;
        }
        getchar();
        getchar();
        printf("Type (1.Enqueue  2.Dequeue  3.Display front element  4.Display all
element  5.exit): \n");
        scanf("%d", &c);

    } while (c != 5);
         return 0;
}
```