```
/* DS Lab 6
Develop a menu driven Program in C for the
following operations on Circular QUEUE of
Characters (Array Implementation of Circular Queue
with maximum size MAX)
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
C. Demonstrate Overflow and Underflow situations on
Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for
each of the above operations
*/


// Write C Program to implement Circular Queue
Operation Dynamically by Passing Parameters

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


int CQ_SIZE = 1; // Set an initial Circular Queue
size

// Function to insert an item into the Circular
Queue at the rear
void insert_rear(char item, char **cqueue, int *rear
, int *count)
{
    // Check for overflow of Circular Queue
    if (*count == CQ_SIZE)
    {
        printf("Circular Queue overflow,
        reallocating memory to Circular Queue to
        store an item...\n");
        CQ_SIZE++;
```

```
            *cqueue = (char *)realloc(*cqueue, CQ_SIZE *
             sizeof(char));

            // Adjust for circular behavior after
            expanding the queue
            if (*rear < *count - 1)
            {
                for (int i = 0; i < *rear + 1; i++)
                    (*cqueue)[CQ_SIZE - *rear - 1 + i] =
                     (*cqueue)[i];

                *rear = CQ_SIZE - 1;
            }
        }

        // Insert an item into the Circular Queue
        *rear = (*rear + 1) % CQ_SIZE;
        (*cqueue)[*rear] = item;
        (*count)++;
}

// Function to delete an item from the front of the
Circular Queue
void delete_front(char *cqueue, int *front, int *
count)
{
        if (*count == 0)
        {
            printf("Circular Queue underflow\n");
            return; // Indicating the Circular queue is
            empty
        }

        printf("Item deleted: %c\n", cqueue[*front]);
        *front = (*front + 1) % CQ_SIZE;
        (*count)--;
}
```

```
// Function to display the elements of the Circular
Queue
void display(char *cqueue, int front, int count)
{
    // Check for empty Circular Queue
    if (count == 0)
    {
        printf("Circular Queue is Empty\n");
        return;
    }

    // Display contents in the Circular Queue
    printf("Circular Queue Elements: ");
    for (int i = 0; i < count; i++)
        printf("%c ", cqueue[(front + i) % CQ_SIZE
        ]);

    printf("\n");
}

void main()
{
    int choice, front = 0, rear = -1, count = 0;
    char item, *cqueue;
    cqueue = (char *)malloc(CQ_SIZE * sizeof(char));

    for (;;)
    {
        printf("1: Insert rear 2: Delete front 3:
        Display 4: Exit : ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter the Character : ");
                scanf(" %c", &item);  // Space
```

```
                    before %c to consume newline

                    insert_rear(item, &cqueue, &rear, &
                    count);
                    break;

                case 2:
                    delete_front(cqueue, &front, &count
                    );
                    break;

                case 3:
                    display(cqueue, front, count);
                    break;

                case 4:
                    free(cqueue); // Free allocated
                    memory
                    exit(0);

                default:
                    printf("Invalid choice. Please try
                    again.\n");
            }
        }
}

/*
Output :

1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue is Empty
1: Insert rear 2: Delete front 3: Display 4: Exit :
2
Circular Queue underflow
1: Insert rear 2: Delete front 3: Display 4: Exit :
1
```

```
Enter the Character : A
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: A
1: Insert rear 2: Delete front 3: Display 4: Exit :
1
Enter the Character : B
Circular Queue overflow, reallocating memory to
Circular Queue to store an item...
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: A B
1: Insert rear 2: Delete front 3: Display 4: Exit :
1
Enter the Character : C
Circular Queue overflow, reallocating memory to
Circular Queue to store an item...
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: A B C
1: Insert rear 2: Delete front 3: Display 4: Exit :
1
Enter the Character : D
Circular Queue overflow, reallocating memory to
Circular Queue to store an item...
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: A B C D
1: Insert rear 2: Delete front 3: Display 4: Exit :
1
Enter the Character : E
Circular Queue overflow, reallocating memory to
Circular Queue to store an item...
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: A B C D E
1: Insert rear 2: Delete front 3: Display 4: Exit :
2
```

```
Item deleted: A
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: B C D E
1: Insert rear 2: Delete front 3: Display 4: Exit :
2
Item deleted: B
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: C D E
1: Insert rear 2: Delete front 3: Display 4: Exit :
2
Item deleted: C
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: D E
1: Insert rear 2: Delete front 3: Display 4: Exit :
2
Item deleted: D
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue Elements: E
1: Insert rear 2: Delete front 3: Display 4: Exit :
2
Item deleted: E
1: Insert rear 2: Delete front 3: Display 4: Exit :
3
Circular Queue is Empty
1: Insert rear 2: Delete front 3: Display 4: Exit :
5
Invalid choice. Please try again.
1: Insert rear 2: Delete front 3: Display 4: Exit :
4


...Program finished with exit code 0
Press ENTER to exit console.
```

```
*/
```