**Practical-6**

**1.** **Implement the following functionalities of the Circular queue using Arrays:**

**a. isFull – to check if the queue is full or not.**

**b. isEmpty – to check if the queue is empty or not.**

**c. enqueue – to insert the element in the queue.**

**d. dequeue – to delete the element from the queue.**

**e. front and rear – to print the front and rear element of the queue.**

Aim:

Implementation of circular queue.

Theory:

In this program we implement circular queue by reseting the values of front and rear according to the situation and we also reference to address to help us save memory.

Code:

#include<stdio.h>

#include<stdlib.h>

struct queue

{

    int f,r,size; // Here r is pointing to next address not the last element address

    int \* array;

};

struct queue \* createQueue(int size)

{

    struct queue \*q=(struct queue \*)malloc(sizeof(struct queue));

    q->f=-1;

    q->r=-1;

    q->size=size;

    q->array=(int \*)malloc(sizeof(int)\*size);

    return q;

};

int isEmpty(struct queue \*q)

{

    if(q->f==-1)

    {

        return 0;

    }

    return 1;

}

int isFull(struct queue \*q)

{

    if(q->r-1==q->size-1 && q->f==0)

    {

        return 0;

    }

    else if(q->f==(q->r-1)% q->size)

    {

        return 0;

    }

    return 1;

}

int Enqueue(struct queue \*q)

{

    int item;

    printf("Enter the number you want to insert=");

    scanf("%d",&item);

    if(isEmpty(q)==0)

    {

        q->f=0;

        q->r=1;

        q->array[q->f]=item;

        printf("Element inserted\n");

    }

    else if(q->r<=q->size-1)

    {

        q->array[q->r]=item;

        q->r++;

    }

    else

    {

        q->r=0;

        q->array[q->r]=item;

        q->r++;

    }

    return 0;

}

int Dequeue(struct queue \*q)

{

    if(isEmpty(q)==0)

    {

        printf("No element to delete/dequeue\n");

    }

    else if(q->f==q->size-1)

    {

        q->f=0;

    }

    else

    {

        printf("The element %d is deleted \n",q->array[q->f]);

        q->f++;

    }

}

int print(struct queue \*q)

{

    printf("The front element is %d\n",q->array[q->f]);

    printf("The rear element is %d\n",q->array[q->r-1]);

}

int main()

{

    int size,i;

    printf("Enter the size for queue=");

    scanf("%d",&size);

    struct queue \*q=createQueue(size);

    printf("Enter a number for the following choice=\n");

    while(1)

    {

        printf("1.To check for empty queue \n2.To check if queue is full \n3.To insert element \n4.To delete element \n5.To print front and rear element \n6.To exit\n");

        scanf("%d",&i);

        switch(i)

        {

            case 1:

                if(isEmpty(q)==0)

                {

                    printf("The queue is empty\n");

                }

                else

                    printf("The queue is not empty\n");

                break;

            case 2:

                if(isFull(q)==0)

                {

                    printf("The queue is full\n");

                }

                else

                    printf("The queue is not full\n");

                break;

            case 3:

                Enqueue(q);

                break;

            case 4:

                Dequeue(q);

                break;

            case 5:

                print(q);

                break;

            case 6:

                exit(0);

                break;

            default:

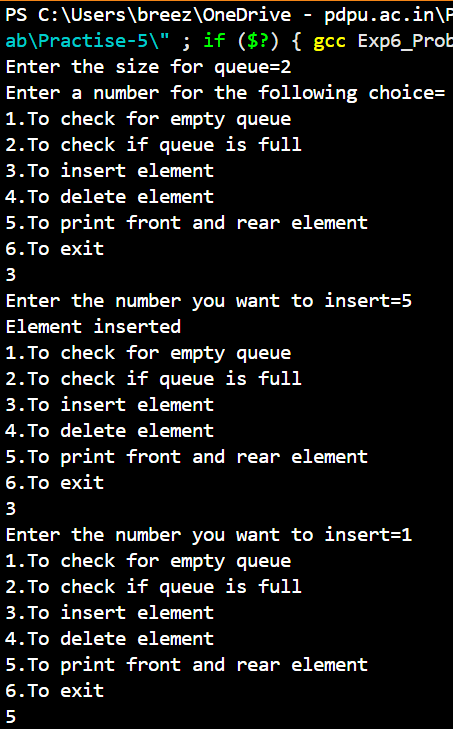
                printf("Invalid input");

        }

    }

}

Output:



A screen shot of a computer

AI-generated content may be incorrect.

**Link for all code:**

<https://github.com/PanavPatel06/DSA-Lab/tree/main/Practise-5>