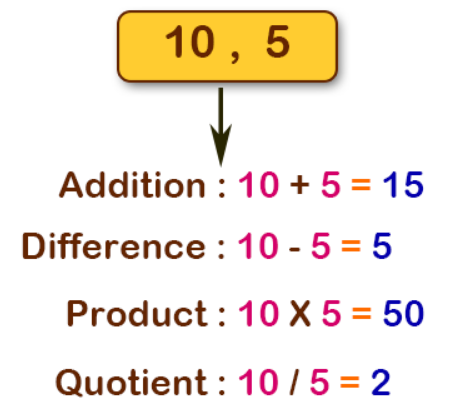
**Experiment-1(functions and arrays)**

**Q1.Write a program to create methods for performing addition, subtraction, multiplication and division on 2 integers.**



**#include<stdio.h>**

**// functions declaration**

**int add(int n1, int n2);**

**int subtract(int n1, int n2);**

**int multiply(int n1, int n2);**

**int divide(int n1, int n2);**

**// main function**

**int main()**

**{**

**int num1, num2;**

**printf("Enter two numbers: ");**

**scanf("%d %d", &num1, &num2);**

**printf("%d + %d = %d\n", num1, num2, add(num1, num2));**

**printf("%d - %d = %d\n", num1, num2, subtract(num1, num2));**

**printf("%d \* %d = %d\n", num1, num2, multiply(num1, num2));**

**printf("%d / %d = %d\n", num1, num2, divide(num1, num2));**

**return 0;**

**}**

**// function to add two integer numbers**

**int add(int n1, int n2)**

**{**

**int result;**

**result = n1 + n2;**

**return result;**

**}**

**// function to subtract two integer numbers**

**int subtract(int n1, int n2)**

**{**

**int result;**

**result = n1 - n2;**

**return result;**

**}**

**// function to multiply two integer numbers**

**int multiply(int n1, int n2)**

**{**

**int result;**

**result = n1 \* n2;**

**return result;**

**}**

**// function to divide two integer numbers**

**int divide(int n1, int n2)**

**{**

**int result;**

**result = n1 / n2;**

**return result;**

**}**

**Output:**

**Enter two numbers: 20 5  
20 + 5 = 25  
20 – 5 = 15  
20 \* 5 = 100  
20 / 5 = 4**

**Q2.Write a program to create an UDF for input 10 numbers into a 1D array. Create two functions MAX() and MIN(). MAX() is used to return the largest element and MIN() is used to return the smallest number in array.**

**#include <stdio.h>**

**#include <conio.h>**

**int sumofarray(int a[],int n)**

**{**

**int min,max,i;**

**min=max=a[0];**

**for(i=1; i<n; i++)**

**{**

**if(min>a[i])**

**min=a[i];**

**if(max<a[i])**

**max=a[i];**

**}**

**printf("minimum of array is : %d",min);**

**printf("\nmaximum of array is : %d",max);**

**}**

**int main()**

**{**

**int a[1000],i,n,sum;**

**printf("Enter size of the array : ");**

**scanf("%d", &n);**

**printf("Enter elements in array : ");**

**for(i=0; i<n; i++)**

**{**

**scanf("%d",&a[i]);**

**}**

**sumofarray(a,n);**

**}**

**Output:**

**Enter size of the array: 5**

**Enter elements in array: 1**

**2**

**35**

**0**

**-1**

**minimum of an array is: -1**

**maximum of an array is: 35**

**Q3) Write a C program to create methods for operations insertion and display on 1D array of elements using UDF.**

**#include <stdio.h>**

**void insertElement(int arr[], int n, int x, int pos)**

**{**

**for (int i = n - 1; i >= pos; i--)**

**arr[i + 1] = arr[i];**

**arr[pos] = x;**

**}**

**int main()**

**{**

**int arr[15] = { 2, 4, 1, 8, 5 };**

**int n = 5;**

**printf("Before insertion : ");**

**for (int i = 0; i < n; i++)**

**printf("%d ", arr[i]);**

**printf("\n");**

**int x = 10, pos = 2;**

**insertElement(arr, n, x, pos);**

**n++;**

**printf("After insertion : ");**

**for (int i = 0; i < n; i++)**

**printf("%d ", arr[i]);**

**return 0;**

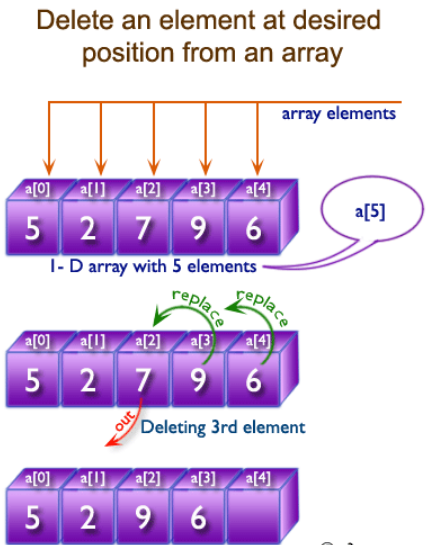
**}**

**Output:**

**Before insertion : 2 4 1 8 5**

**After insertion : 2 4 10 1** 8 5

**Q4) Write a C program to create methods for operations deletion, and display on 1D array of elements using UDF.**

****

**#include <stdio.h>**

**int findElement(int arr[], int n, int key);**

**int deleteElement(int arr[], int n, int key)**

**{**

**// Find position of element to be deleted**

**int pos = findElement(arr, n, key);**

**if (pos == -1)**

**{**

**printf("Element not found");**

**return n;**

**}**

**int i;**

**for (i = pos; i < n - 1; i++)**

**arr[i] = arr[i + 1];**

**return n - 1;**

**}**

**int findElement(int arr[], int n, int key)**

**{**

**int i;**

**for (i = 0; i < n; i++)**

**if (arr[i] == key)**

**return i;**

**return -1;**

**}**

**int main()**

**{**

**int i;**

**int arr[] = { 10, 50, 30, 40, 20 };**

**int n = sizeof(arr) / sizeof(arr[0]);**

**int key = 30;**

**printf("Array before deletion\n");**

**for (i = 0; i < n; i++)**

**printf("%d ", arr[i]);**

**n = deleteElement(arr, n, key);**

**printf("\nArray after deletion\n");**

**for (i = 0; i < n; i++)**

**printf("%d ", arr[i]);**

**return 0;**

**}**

**Output:**

**Array before deletion**

**10 50 30 40 20**

**Array after deletion**

**10 50 40 20**

**Experiment 2: (concepts of matrix and sparse matrix)**

**Q1) Write a C program to create function for performing matrix multiplication using UDF**

* This C program is to multiply two matrices using function.For example, for a 2 x 2 matrix, the **multiplication** of two matrices matrix1 {1,2,3,4} and matrix2 {5,6,7,8} will be equal to **mat{19,22,43,50}.**

**1     2                      5       6                       19        22**

    X                                 =

**3    4                       7       8                       43      50**

**#include<stdio.h>**

**void multiply(int mat1[12][12],int mat2[12][12],int ,int ,int );**

**void main()**

**{**

**int mat1[12][12],mat2[12][12];**

**int i,j,k,m,n,p;**

**printf("Enter the number of rows and columns for 1st matrix\n");**

**scanf("%d%d",&m,&n);**

**printf("Enter the elements of the 1st matrix\n");**

**for(i=0;i<m;i++)**

**{**

**for(j=0;j<n;j++)**

**{**

**scanf("%d",&mat1[i][j]);**

**}**

**}**

**//no of col of 1st mat = no of rows of 2nd mat**

**printf("Enter the number of columns for 2nd matrix\n");**

**scanf("%d",&p);**

**printf("Enter the elements of the 2nd matrix\n");**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<p;j++)**

**{**

**scanf("%d",&mat2[i][j]);**

**}**

**}**

**printf("The 1st matrix\n");**

**for(i=0;i<m;i++)**

**{**

**for(j=0;j<n;j++)**

**{**

**printf("%d\t",mat1[i][j]);**

**}**

**printf("\n");**

**}**

**printf("The 2nd matrix\n");**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<p;j++)**

**{**

**printf("%d\t",mat2[i][j]);**

**}**

**printf("\n");**

**}**

**multiply(mat1,mat2,m,n,p);**

**}**

**void multiply(int mat1[12][12],int mat2[12][12],int m,int n,int p)**

**{**

**int mul[12][12],i,j,k;**

**for(i=0;i<m;i++)**

**{**

**for(j=0;j<p;j++)**

**{**

**mul[i][j]=0;**

**for(k=0;k<n;k++)**

**{**

**mul[i][j]=mul[i][j]+mat1[i][k]\*mat2[k][j];**

**}**

**}**

**}**

**printf("The resultant matrix formed on multiplying the two matrices\n");**

**for(i=0;i<m;i++)**

**{**

**for(j=0;j<p;j++)**

**{**

**printf("%d\t",mul[i][j]);**

**}**

**printf("\n");**

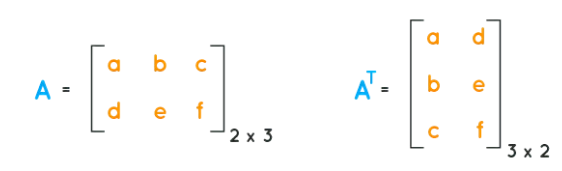
**}**

**}**

**Q2) Write a C program to input elements into a square matrix and display the transpose of it using UDF.**

**What is the Transpose of a Matrix?**

* The **transpose of a matrix** is obtained by changing its rows into columns (or equivalently, its columns into rows). A rectangular array of numbers or functions that are arranged in the form of rows and columns is called a [matrix](https://www.cuemath.com/algebra/solve-matrices/). This array of numbers are called either entries or elements of a matrix.
* Here for matrix A the elements of the first row have been written in the first column of the new matrix, and the elements of the second row have been written in the second column of the new matrix. And this new matrix is denoted as AT, which is the transpose of the given matrix A.



**#include<stdio.h>**

**void transpose(float a[10][10], int n);**

**int main()**

**{**

**int i,j,n;**

**float a[10][10];**

**printf("Enter order of matrix:\n");**

**scanf("%d", &n);**

**printf("Enter matrix elements:\n");**

**for(i=0;i< n;i++)**

**{**

**for(j=0;j< n;j++)**

**{**

**printf("a[%d][%d]=",i,j);**

**scanf("%f", &a[i][j]);**

**}**

**}**

**transpose(a,n);**

**for(i=0;i< n;i++)**

**{**

**for(j=0;j< n;j++)**

**{**

**printf("%f\t",a[i][j]);**

**}**

**printf("\n");**

**}**

**return 0;**

**}**

**void transpose(float a[10][10], int n)**

**{**

**int i,j;**

**float tmp;**

**for(i=0;i< n-1;i++)**

**{**

**for(j=i+1;j< n;j++)**

**{**

**tmp = a[i][j];**

**a[i][j] = a[j][i];**

**a[j][i] = tmp;**

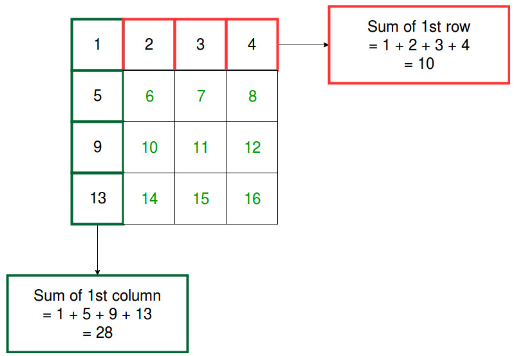
**}**

**}**

**}**

**Q3) Write a program to input elements into a 4X4 matrix and display the sum of individual row elements using UDF.**

**Approach:**

****

**/\* C Program to find Sum of rows in a Matrix \*/**

**#include<stdio.h>**

**void addRows(int arr[10][10], int i, int j)**

**{**

**int rows, columns;**

**for(rows = 0; rows < i; rows++)**

**{**

**int Sum = 0;**

**for(columns = 0; columns < j; columns++)**

**{**

**Sum = Sum + arr[rows][columns];**

**}**

**printf("The Sum of Elements of a Rows in a Matrix = %d \n", Sum );**

**}**

**}**

**int main()**

**{**

**int i, j, rows, columns, a[10][10];**

**printf("Please Enter Number of rows and columns : ");**

**scanf("%d %d", &i, &j);**

**printf("Please Enter the Matrix Row and Column Elements \n");**

**for(rows = 0; rows < i; rows++)**

**{**

**for(columns = 0; columns < j; columns++)**

**{**

**scanf("%d", &a[rows][columns]);**

**}**

**}**

**addRows(a, i, j);**

**return 0;**

**}**

**Output:**

**Please Enter Number of rows and columns : 3 3**

**Please Enter the Matrix Row and Column Elements**

**10 20 30**

**40 50 760**

**70 80 90**

**The Sum of Elements of a Rows in a Matrix = 60**

**The Sum of Elements of a Rows in a Matrix = 850**

**The Sum of Elements of a Rows in a Matrix = 240**

**Q4) Write a program to input elements into a 4X4 matrix, check it for sparse or not. If sparse then store the non-zero elements into an alternate matrix and then display it using UDF.**

**#include<stdio.h>**

**void sparse(int arr[10][10]);**

**void new\_matrix(int arr\_new[10][10]);**

**int a,b,i,j;**

**int main()**

**{**

**printf("Enter the number of rows and columns:");**

**scanf("%d %d",&a,&b);**

**int arr[a][b];**

**sparse(arr);**

**return 0;**

**}**

**void sparse(int arr[a][b])**

**{**

**printf("enter the matrix:");**

**for(i=0;i<a;i++)**

**{**

**for(j=0;j<b;j++)**

**{**

**scanf("%d",&arr[i][j]);**

**}**

**printf("\n");**

**}**

**printf("The matrix is:\n");**

**for(i=0;i<a;i++)**

**{**

**for(j=0;j<b;j++)**

**{**

**printf("%d",arr[i][j]);**

**}**

**printf("\n");**

**}**

**int m=a\*b;**

**int k=0;**

**printf("The total number of elements you have entered is:%d\n",m);**

**for(i=0;i<a;i++)**

**{**

**for(j=0;j<b;j++)**

**{**

**if(arr[i][j]==0)**

**{**

**k++;**

**}**

**}**

**}**

**if(k>m/2)**

**{**

**printf("The matrix is a sparse matrix");**

**}**

**else**

**printf("The matrix is not a sparse matrix");**

**}**

**Sample Output:**

**Enter the number of rows and columns:5**

**5**

**enter the matrix:1**

**2**

**5**

**6**

**0**

**5**

**0**

**1**

**0**

**0**

**0**

**7**

**0**

**8**

**0**

**9**

**0**

**3**

**0**

**20**

**0**

**0**

**0**

**0**

**0**

**The matrix is:**

**12560**

**50100**

**07080**

**903020**

**00000**

**The total number of elements you have entered is:25**

**The matrix is a sparse matrix**

**--------------------------------**

**Process exited after 24.79 seconds with return value 0**

**Press any key to continue . . .**

**Experiment 3: (stack and queue)**

**Q1) Write a program using C to create a stack of numbers and perform using UDF:**

**(i) push operation (ii) pop operation (iii) display operation**

**#include<stdio.h>**

**int stack[100],choice,n,top,x,i;**

**void push(void);**

**void pop(void);**

**void display(void);**

**int main()**

**{**

**top=-1;**

**printf("\n Enter the size of STACK[MAX=100]:");**

**scanf("%d",&n);**

**printf("\n\t STACK OPERATIONS USING ARRAY");**

**printf("\n\t--------------------------------");**

**printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");**

**do**

**{**

**printf("\n Enter the Choice:");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:**

**{**

**push();**

**break;**

**}**

**case 2:**

**{**

**pop();**

**break;**

**}**

**case 3:**

**{**

**display();**

**break;**

**}**

**case 4:**

**{**

**printf("\n\t EXIT POINT ");**

**break;**

**}**

**default:**

**{**

**printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");**

**}**

**}**

**}**

**while(choice!=4);**

**return 0;**

**}**

**void push()**

**{**

**if(top>=n-1)**

**{**

**printf("\n\tSTACK is over flow");**

**}**

**else**

**{**

**printf(" Enter a value to be pushed:");**

**scanf("%d",&x);**

**top++;**

**stack[top]=x;**

**}**

**}**

**void pop()**

**{**

**if(top<=-1)**

**{**

**printf("\n\t Stack is under flow");**

**}**

**else**

**{**

**printf("\n\t The popped elements is %d",stack[top]);**

**top--;**

**}**

**}**

**void display()**

**{**

**if(top>=0)**

**{**

**printf("\n The elements in STACK \n");**

**for(i=top; i>=0; i--)**

**printf("\n%d",stack[i]);**

**printf("\n Press Next Choice");**

**}**

**else**

**{**

**printf("\n The STACK is empty");**

**}**

**}**

**Sample Output:**

**Enter the size of STACK[MAX=100]:10**

**STACK OPERATIONS USING ARRAY**

**--------------------------------**

**1.PUSH**

**2.POP**

**3.DISPLAY**

**4.EXIT**

**Enter the Choice:1**

**Enter a value to be pushed:12**

**Enter the Choice:1**

**Enter a value to be pushed:24**

**Enter the Choice:1**

**Enter a value to be pushed:98**

**Enter the Choice:3**

**The elements in STACK**

**98**

**24**

**12**

**Press Next Choice**

**Enter the Choice:2**

**The popped elements is 98**

**Enter the Choice:3**

**The elements in STACK**

**24**

**12**

**Press Next Choice**

**Enter the Choice:4**

**EXIT POINT**

**Q2) Write a C program to create a linear queue and perform the following operations using UDF: (i) insertion ii) deletion and iii) Traversal**

**#include <stdio.h>**

**# define SIZE 100**

**void enqueue();**

**void dequeue();**

**void show();**

**int inp\_arr[SIZE];**

**int Rear = - 1;**

**int Front = - 1;**

**main()**

**{**

**int ch;**

**while (1)**

**{**

**printf("1.Enqueue Operation\n");**

**printf("2.Dequeue Operation\n");**

**printf("3.Display the Queue\n");**

**printf("4.Exit\n");**

**printf("Enter your choice of operations : ");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**enqueue();**

**break;**

**case 2:**

**dequeue();**

**break;**

**case 3:**

**show();**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("Incorrect choice \n");**

**}**

**}**

**}**

**void enqueue()**

**{**

**int insert\_item;**

**if (Rear == SIZE - 1)**

**printf("Overflow \n");**

**else**

**{**

**if (Front == - 1)**

**Front = 0;**

**printf("Element to be inserted in the Queue\n : ");**

**scanf("%d", &insert\_item);**

**Rear = Rear + 1;**

**inp\_arr[Rear] = insert\_item;**

**}**

**}**

**void dequeue()**

**{**

**if (Front == - 1 || Front > Rear)**

**{**

**printf("Underflow \n");**

**return ;**

**}**

**else**

**{**

**printf("Element deleted from the Queue: %d\n", inp\_arr[Front]);**

**Front = Front + 1;**

**}**

**}**

**void show()**

**{**

**if (Front == - 1)**

**printf("Empty Queue \n");**

**else**

**{**

**printf("Queue: \n");**

**for (int i = Front; i <= Rear; i++)**

**printf("%d ", inp\_arr[i]);**

**printf("\n");**

**}**

**}**

**Sample Output:**

**1.Enqueue Operation**

**2.Dequeue Operation**

**3.Display the Queue**

**4.Exit**

**Enter your choice of operations : 1**

**Element to be inserted in the Queue: 10**

**1.Enqueue Operation**

**2.Dequeue Operation**

**3.Display the Queue**

**4.Exit**

**Enter your choice of operations : 1**

**Element to be inserted in the Queue: 20**

**1.Enqueue Operation**

**2.Dequeue Operation**

**3.Display the Queue**

**4.Exit**

**Enter your choice of operations : 3**

**Queue:**

**10 20**

**1.Enqueue Operation**

**2.Dequeue Operation**

**3.Display the Queue**

**4.Exit**

**Enter your choice of operations : 2**

**Element deleted from the Queue: 10**

**1.Enqueue Operation**

**2.Dequeue Operation**

**3.Display the Queue**

**4.Exit**

**Enter your choice of operations: 3**

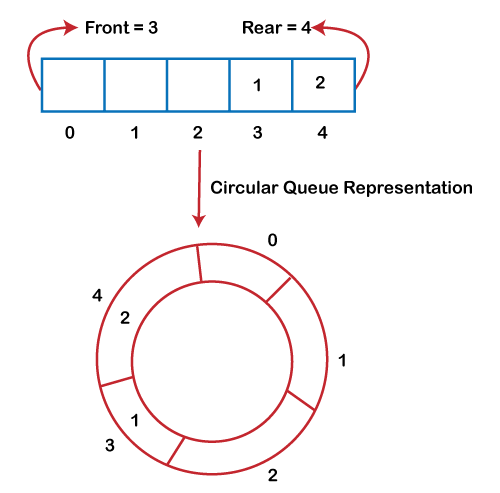
**Queue:**

**20**

**Q3) Write a C program to create a circular queue and perform the following operations using UDF: (i) insertion ii) deletion and iii) Traversal**

# **Circular Queue**

* There was one limitation in the array implementation of [Queue](https://www.javatpoint.com/data-structure-queue). If the rear reaches to the end position of the Queue then there might be possibility that some vacant spaces are left in the beginning which cannot be utilized. So, to overcome such limitations, the concept of the circular queue was introduced.



* A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a **Ring Buffer**.

### **Operations on Circular Queue**

The following are the operations that can be performed on a circular queue:

* **Front:** It is used to get the front element from the Queue.
* **Rear:** It is used to get the rear element from the Queue.
* **enQueue(value):** This function is used to insert the new value in the Queue. The new element is always inserted from the rear end.
* **deQueue():** This function deletes an element from the Queue. The deletion in a Queue always takes place from the front end.

### **Applications of Circular Queue**

**The circular Queue can be used in the following scenarios:**

* **Memory management:** The circular queue provides memory management. As we have already seen that in linear queue, the memory is not managed very efficiently. But in case of a circular queue, the memory is managed efficiently by placing the elements in a location which is unused.
* **CPU Scheduling:** The operating system also uses the circular queue to insert the processes and then execute them.
* **Traffic system:** In a computer-control traffic system, traffic light is one of the best examples of the circular queue. Each light of traffic light gets ON one by one after every jinterval of time. Like red light gets ON for one minute then yellow light for one minute and then green light. After green light, the red light gets ON.

### **Enqueue operation**

**The steps of enqueue operation are given below:**

* First, we will check whether the Queue is full or not.
* Initially the front and rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.
* When we insert a new element, the rear gets incremented, i.e., **rear=rear+1**.

### **Scenarios for inserting an element**

**There are two scenarios in which queue is not full:**

* **If rear != max - 1**, then rear will be incremented to **mod(maxsize)** and the new value will be inserted at the rear end of the queue.
* **If front != 0 and rear = max - 1**, it means that queue is not full, then set the value of rear to 0 and insert the new element there.

**There are two cases in which the element cannot be inserted:**

* When **front ==0** && **rear = max-1**, which means that front is at the first position of the Queue and rear is at the last position of the Queue.
* front== rear + 1;

**Algorithm to insert an element in a circular queue**

**Step 1:** IF (REAR+1)%MAX = FRONT  
Write " OVERFLOW "  
Goto step 4  
[End OF IF]

**Step 2:** IF FRONT = -1 and REAR = -1  
SET FRONT = REAR = 0  
ELSE IF REAR = MAX - 1 and FRONT ! = 0  
SET REAR = 0  
ELSE  
SET REAR = (REAR + 1) % MAX  
[END OF IF]

**Step 3:** SET QUEUE[REAR] = VAL

**Step 4:** EXIT

### **Dequeue Operation**

The steps of dequeue operation are given below:

* First, we check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.
* When the element is deleted, the value of front gets decremented by 1.
* If there is only one element left which is to be deleted, then the front and rear are reset to -1.

**Algorithm to delete an element from the circular queue**

**Step 1:** IF FRONT = -1  
Write " UNDERFLOW "  
Goto Step 4  
[END of IF]

**Step 2:** SET VAL = QUEUE[FRONT]

**Step 3:** IF FRONT = REAR  
SET FRONT = REAR = -1  
ELSE  
IF FRONT = MAX -1  
SET FRONT = 0  
ELSE  
SET FRONT = FRONT + 1  
[END of IF]  
[END OF IF]

**Step 4:** EXIT

**Program:**

**// Circular Queue implementation in C**

**#include <stdio.h>**

**#define SIZE 5**

**int items[SIZE];**

**int front = -1, rear = -1;**

**// Check if the queue is full**

**int isFull()**

**{**

**if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;**

**return 0;**

**}**

**// Check if the queue is empty**

**int isEmpty()**

**{**

**if (front == -1) return 1;**

**return 0;**

**}**

**// Adding an element**

**void enQueue(int element)**

**{**

**if (isFull())**

**printf("\n Queue is full!! \n");**

**else {**

**if (front == -1) front = 0;**

**rear = (rear + 1) % SIZE;**

**items[rear] = element;**

**printf("\n Inserted -> %d", element);**

**}**

**}**

**// Removing an element**

**int deQueue()**

**{**

**int element;**

**if (isEmpty())**

**{**

**printf("\n Queue is empty !! \n");**

**return (-1);**

**} else {**

**element = items[front];**

**if (front == rear)**

**{**

**front = -1;**

**rear = -1;**

**}**

**// Q has only one element, so we reset the**

**// queue after dequeing it. ?**

**else {**

**front = (front + 1) % SIZE;**

**}**

**printf("\n Deleted element -> %d \n", element);**

**return (element);**

**}**

**}**

**// Display the queue**

**void display()**

**{**

**int i;**

**if (isEmpty())**

**printf(" \n Empty Queue\n");**

**else**

**{**

**printf("\n Front -> %d ", front);**

**printf("\n Items -> ");**

**for (i = front; i != rear; i = (i + 1) % SIZE) {**

**printf("%d ", items[i]);**

**}**

**printf("%d ", items[i]);**

**printf("\n Rear -> %d \n", rear);**

**}**

**}**

**int main()**

**{**

**// Fails because front = -1**

**deQueue();**

**enQueue(1);**

**enQueue(2);**

**enQueue(3);**

**enQueue(4);**

**enQueue(5);**

**// Fails to enqueue because front == 0 && rear == SIZE - 1**

**enQueue(6);**

**display();**

**deQueue();**

**display();**

**enQueue(7);**

**display();**

**// Fails to enqueue because front == rear + 1**

**enQueue(8);**

**return 0;**

**}**