Name- Harshita Totala Roll No-SEITA14

Div- A

Assignment-3

Circular Queue

AIM : Implement Circular Queue using Array. Perform following operations on it.

a) Insertion (Enqueue)

b) Deletion (Dequeue)

c) Display

Objectives :

1. To understand the concept of Circular Queue using Array as a data structure.
2. Applications of Circular Queue.

Theory :

1)Definition of Circular Queue –

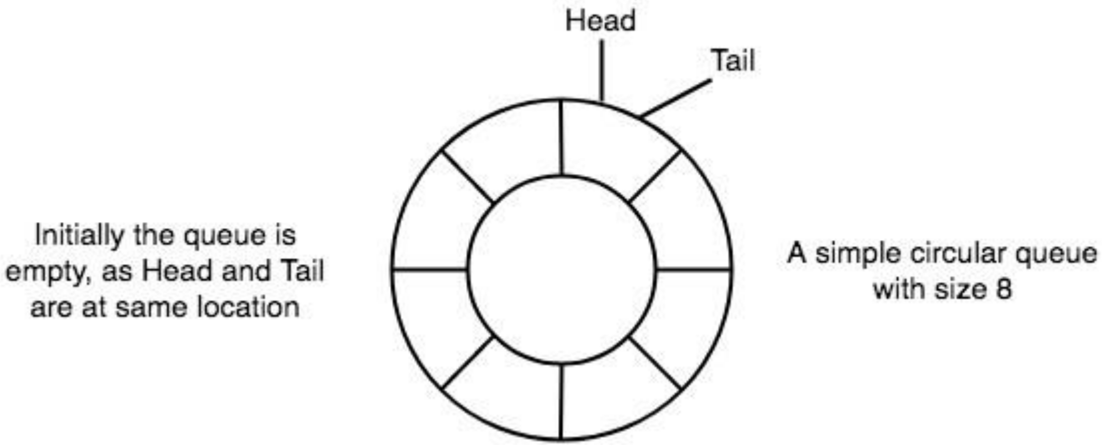
Circular Queue is a linear data structure, which follows the principle of **FIFO**(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

2) Definition of Array –

An **array**, is a data structure consisting of a collection of *elements* (values or variables), each identified by at least one *array index* or *key* .

Basic features of Circular Queue

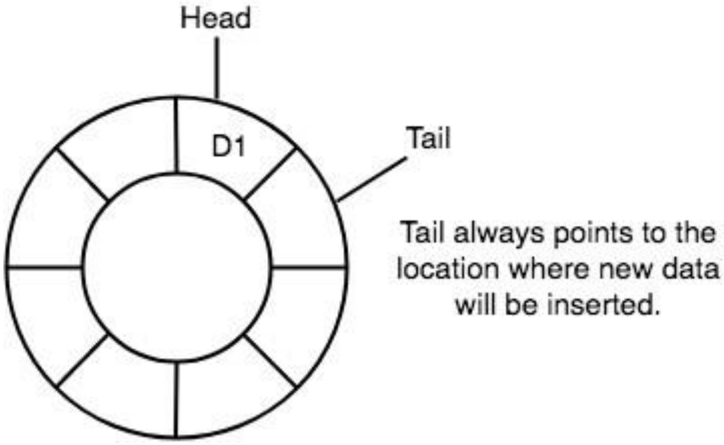
1. In case of a circular queue, `head` pointer will always point to the front of the queue, and `tail` pointer will always point to the end of the queue.
2. Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty.



Initially the queue is empty, as Head and Tail are at same location

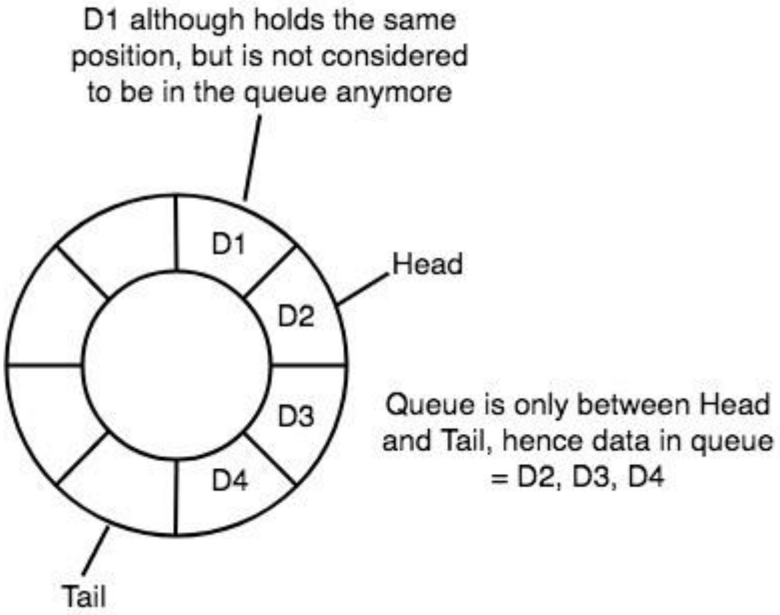
A simple circular queue with size 8

3. New data is always added to the location pointed by the `tail` pointer, and once the data is added, `tail` pointer is incremented to point to the next available location.

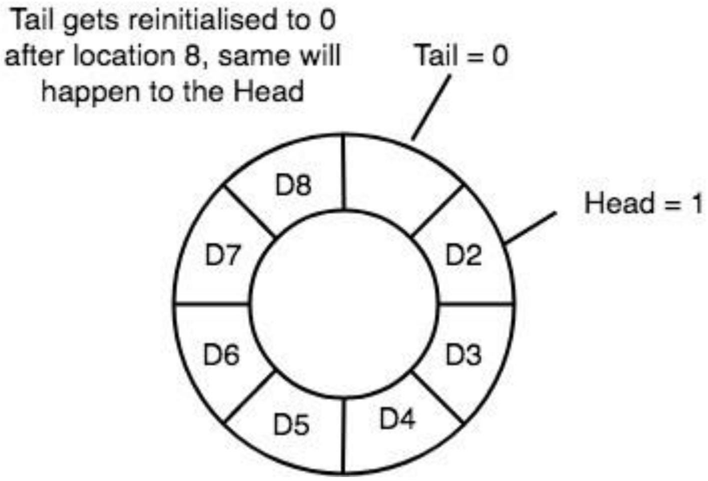


Tail always points to the location where new data will be inserted.

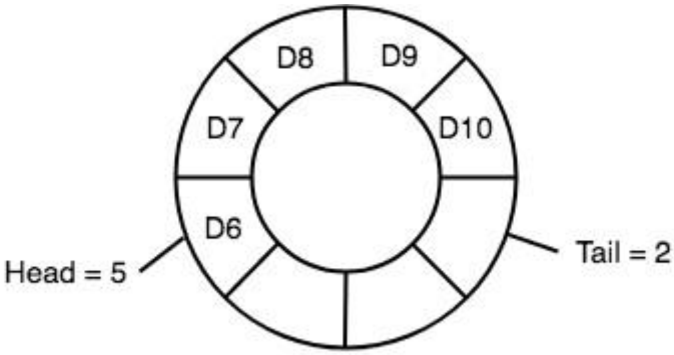
4. In a circular queue, data is not actually removed from the queue. Only the `head` pointer is incremented by one position when **dequeue** is executed. As the queue data is only the data between `head` and `tail`, hence the data left outside is not a part of the queue anymore, hence removed.


D1 although holds the same position, but is not considered to be in the queue anymore
D1
Head
D2
D3
Queue is only between Head and Tail, hence data in queue = D2, D3, D4
D4
Tail

5. The `head` and the `tail` pointer will get reinitialised to **0** every time they reach the end of the queue.


Tail gets reinitialised to 0 after location 8, same will happen to the Head
Tail = 0
D8
Head = 1
D7
D2
D6
D3
D5
D4

6. Also, the `head` and the `tail` pointers can cross each other. In other words, `head` pointer can be greater than the `tail`. Sounds odd? This will happen when we dequeue the queue a couple of times and the `tail` pointer gets reinitialised upon reaching the end of the queue.



In such a situation the value of the Head pointer will be greater than the Tail pointer

Insertion :

enQueue(value) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

Steps:

1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.

Pseudo Code :

```
      void insertCQ(int val)
{     if ((front == 0 && rear == n - 1) || (front == rear +
1))
    {         cout << "Queue
Overflow \n";         return;
    }     if (front
== -1)
    {
front = 0;
rear = 0;
    }     else     {
if (rear == n - 1)
rear = 0;         else
rear = rear + 1;     }
cqueue[rear] = val;
}
```

Deletion :

deQueue() This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

Steps:

1. Check whether queue is Empty means check (front==-1).
2. If it is empty then display Queue is empty. If queue is not empty then step 3
3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

Pseudo Code :

```
      void deleteCQ()

{      if (front == -
1)
    {          cout << "Queue Underflow\n";          return;     }
cout << "Element deleted from queue is : " << cqueue[front] << endl;
if (front == rear)
    {          front
= -1;          rear
= -1;
    }     else     {
if (front == n - 1)
front = 0;          else
front = front + 1;
    } }
```

Display Circular Queue :

We can use the following steps to display the elements of a circular queue...

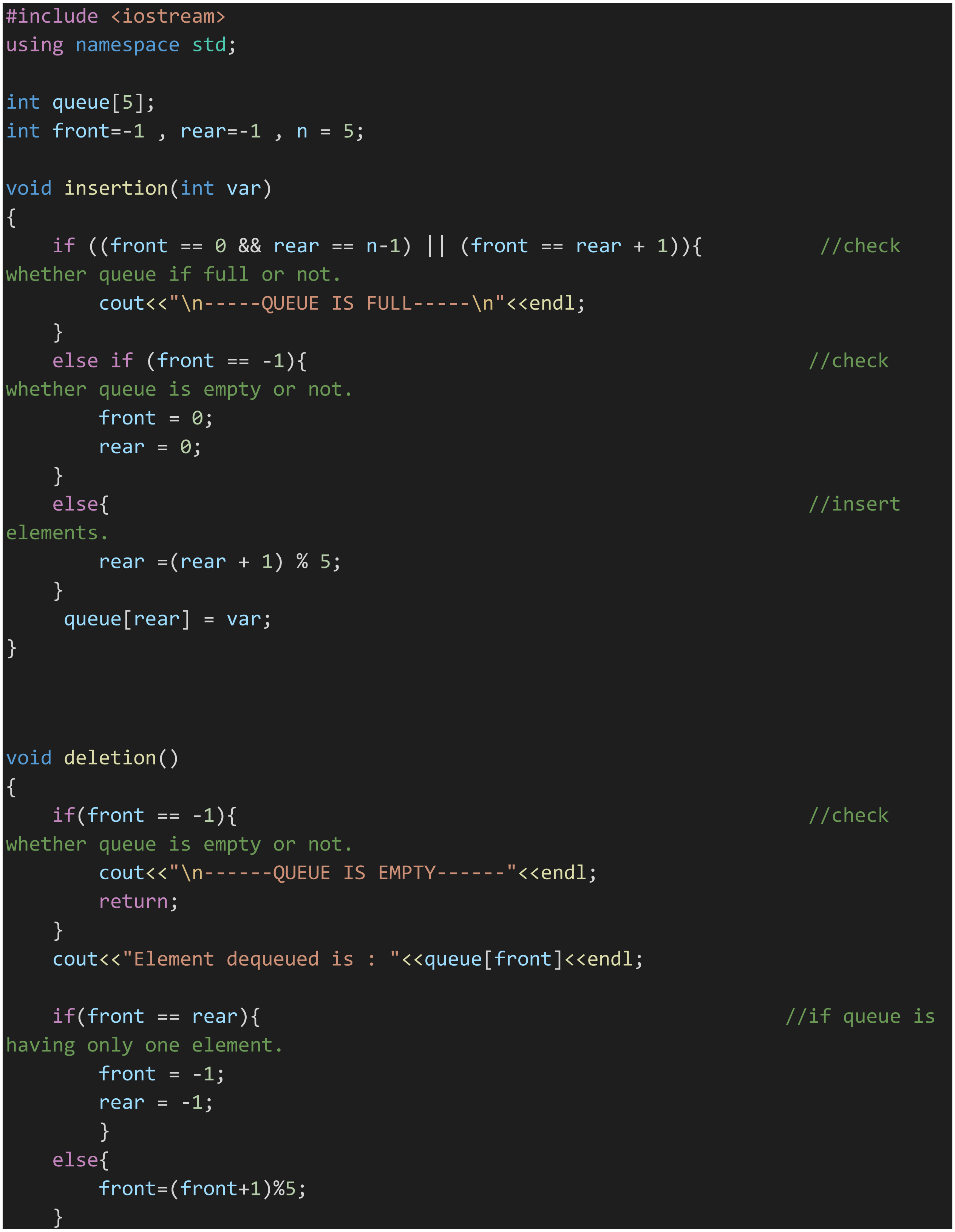
- **Step 1 -** Check whether **queue** is **EMPTY**. (**front == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front**'.
- **Step 4 -** Check whether '**front <= rear**', if it is **TRUE**, then display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i <= rear**' becomes **FALSE**.
- **Step 5 -** If '**front <= rear**' is **FALSE**, then display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until'**i <= SIZE - 1**' becomes **FALSE**.
- **Step 6 -** Set **i** to **0**.
- **Step 7 -** Again display '**cQueue[i]**' value and increment **i** value by one (**i++**). Repeat the same until '**i <= rear**' becomes **FALSE**.

Pseudo Code :

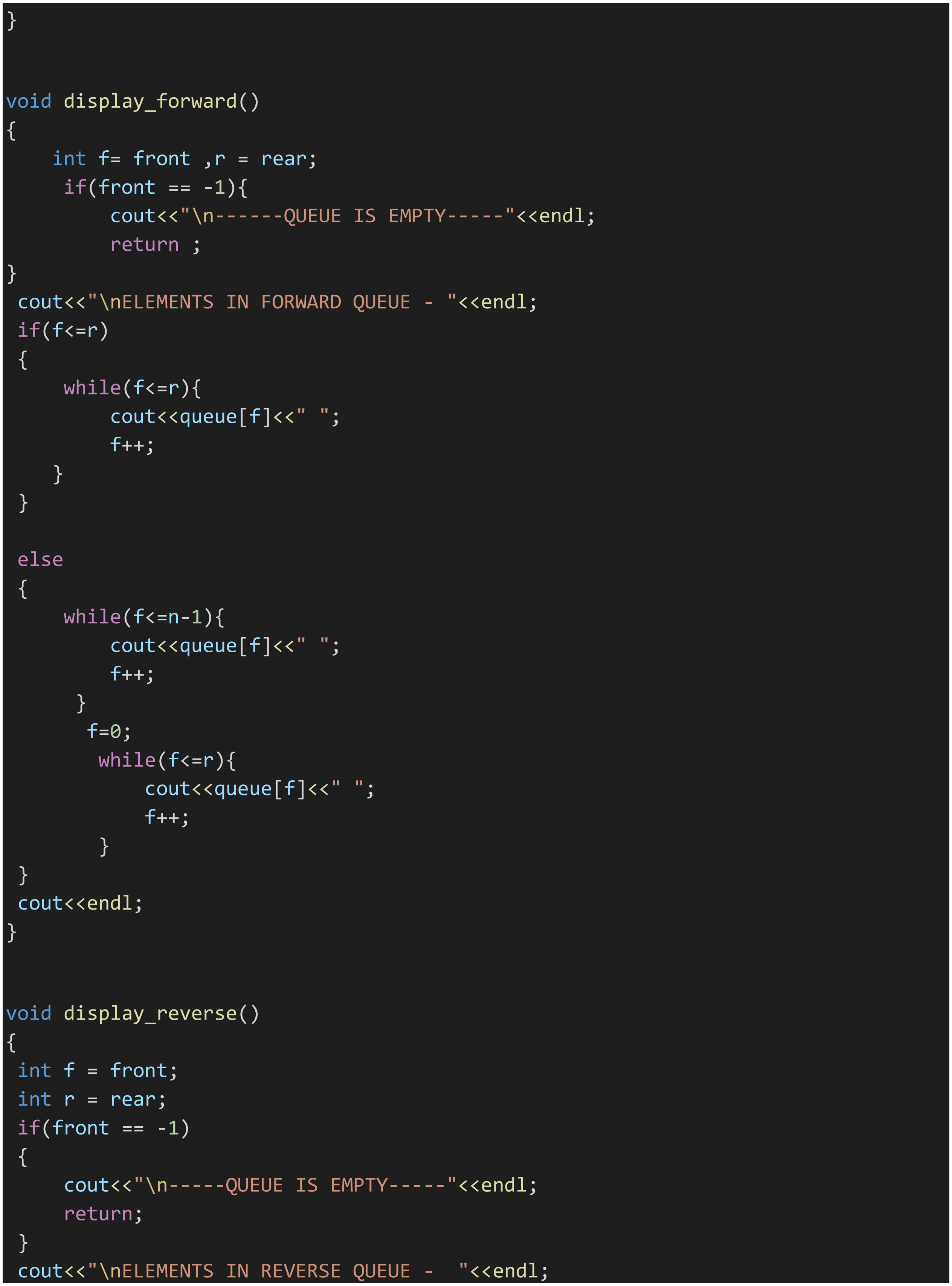
```
      void displayCQ_forward()
```

```
{     int f = front, r =
rear;     if (front == -1)
    {         cout << "Queue is empty" <<
endl;         return;
    }     cout << "Queue elements are
:\n";     if (f <= r)
    {         while (f
<= r)
        {             cout <<
cqueue[f] << " ";             f++;
        }     }     else
{         while (f <= n -
1)
        {             cout <<
cqueue[f] << " ";             f++;
}         f = 0;         while (f <=
r)
        {             cout <<
cqueue[f] << " ";             f++;
        }
}
    cout << endl;
}
```

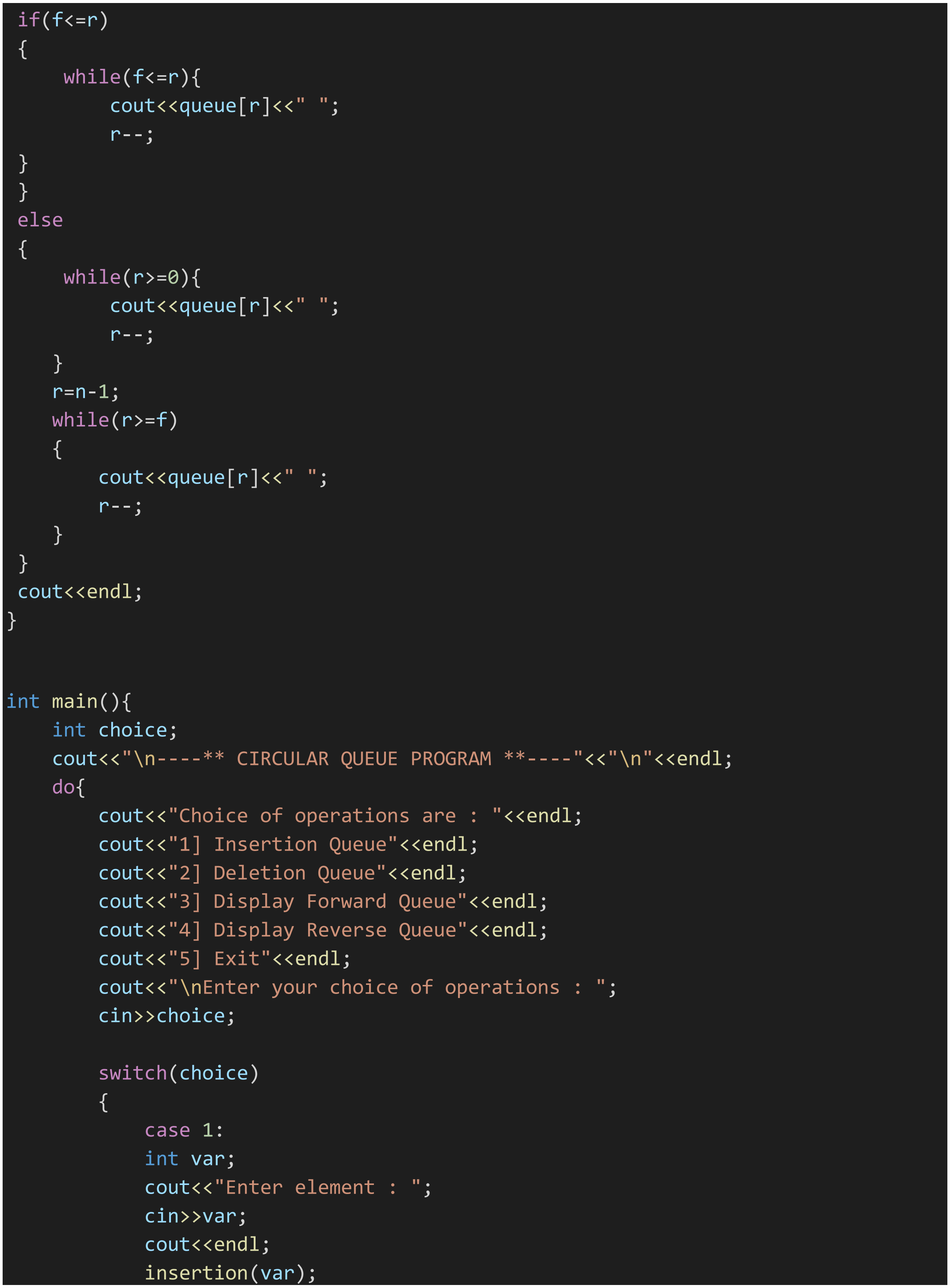
Program-



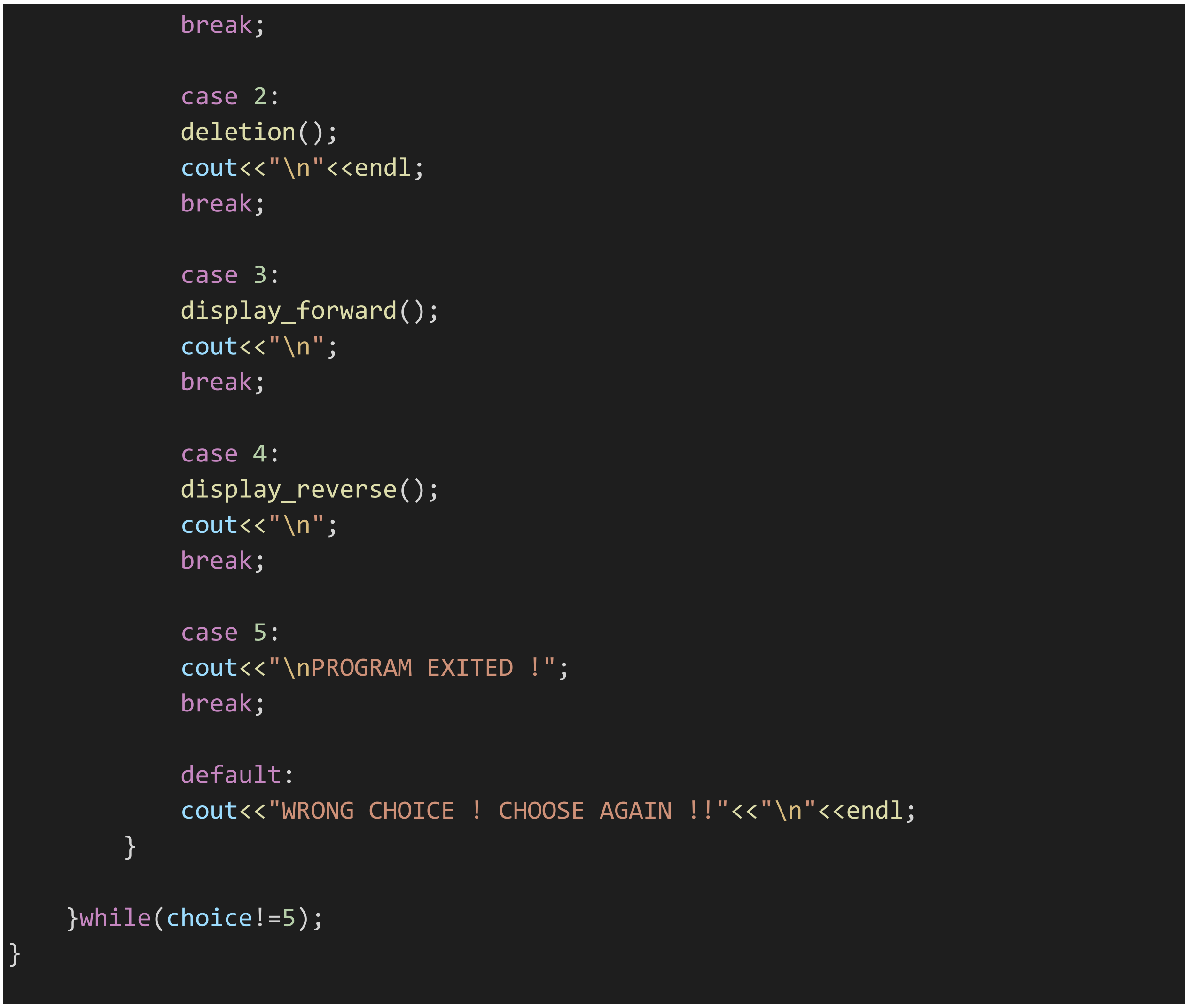
```
#include <iostream>
using namespace std;

int queue[5];
int front=-1 , rear=-1 , n = 5;

void insertion(int var)
{
    if ((front == 0 && rear == n-1) || (front == rear + 1)){          //check
whether queue if full or not.
        cout<<"\n-----QUEUE IS FULL-----\n"<<endl;
    }
    else if (front == -1){                                           //check
whether queue is empty or not.
        front = 0;
        rear = 0;
    }
    else{                                                            //insert
elements.
        rear =(rear + 1) % 5;
    }
     queue[rear] = var;
}



void deletion()
{
    if(front == -1){                                                 //check
whether queue is empty or not.
        cout<<"\n------QUEUE IS EMPTY------"<<endl;
        return;
    }
    cout<<"Element dequeued is : "<<queue[front]<<endl;

    if(front == rear){                                             //if queue is
having only one element.
        front = -1;
        rear = -1;
        }
    else{
        front=(front+1)%5;
    }
```



```
}


void display_forward()
{
    int f= front ,r = rear;
     if(front == -1){
         cout<<"\n------QUEUE IS EMPTY-----"<<endl;
         return ;
}
 cout<<"\nELEMENTS IN FORWARD QUEUE - "<<endl;
 if(f<=r)
 {
     while(f<=r){
         cout<<queue[f]<<" ";
         f++;
    }
 }

 else
 {
     while(f<=n-1){
         cout<<queue[f]<<" ";
         f++;
      }
       f=0;
        while(f<=r){
            cout<<queue[f]<<" ";
            f++;
        }
 }
 cout<<endl;
}


void display_reverse()
{
 int f = front;
 int r = rear;
 if(front == -1)
 {
     cout<<"\n-----QUEUE IS EMPTY-----"<<endl;
     return;
 }
 cout<<"\nELEMENTS IN REVERSE QUEUE -  "<<endl;
```



```
 if(f<=r)
 {
     while(f<=r){
         cout<<queue[r]<<" ";
         r--;
 }
 }
 else
 {
     while(r>=0){
         cout<<queue[r]<<" ";
         r--;
    }
    r=n-1;
    while(r>=f)
    {
        cout<<queue[r]<<" ";
        r--;
    }
 }
 cout<<endl;
}


int main(){
    int choice;
    cout<<"\n----** CIRCULAR QUEUE PROGRAM **----"<<"\n"<<endl;
    do{
        cout<<"Choice of operations are : "<<endl;
        cout<<"1] Insertion Queue"<<endl;
        cout<<"2] Deletion Queue"<<endl;
        cout<<"3] Display Forward Queue"<<endl;
        cout<<"4] Display Reverse Queue"<<endl;
        cout<<"5] Exit"<<endl;
        cout<<"\nEnter your choice of operations : ";
        cin>>choice;

        switch(choice)
        {
            case 1:
            int var;
            cout<<"Enter element : ";
            cin>>var;
            cout<<endl;
            insertion(var);
```



```
            break;

            case 2:
            deletion();
            cout<<"\n"<<endl;
            break;

            case 3:
            display_forward();
            cout<<"\n";
            break;

            case 4:
            display_reverse();
            cout<<"\n";
            break;

            case 5:
            cout<<"\nPROGRAM EXITED !";
            break;

            default:
            cout<<"WRONG CHOICE ! CHOOSE AGAIN !!"<<"\n"<<endl;
        }

    }while(choice!=5);
}
```

Output-

```
----** CIRCULAR QUEUE PROGRAM **----


Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit


Enter your choice of operations : 1

Enter element : 4


Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit
```

```
Enter your choice of operations : 1

Enter element : 7


Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit


Enter your choice of operations : 1

Enter element : 8


Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit


Enter your choice of operations : 3
```

```
ELEMENTS IN FORWARD QUEUE -

4 7 8


Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit


Enter your choice of operations : 4


ELEMENTS IN REVERSE QUEUE -

8 7 4


Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue
```

```
5] Exit


Enter your choice of operations : 2

Element dequeued is : 4




Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit


Enter your choice of operations : 3


ELEMENTS IN FORWARD QUEUE -

7 8


Choice of operations are :

1] Insertion Queue

2] Deletion Queue
```

```
3] Display Forward Queue

4] Display Reverse Queue

5] Exit


Enter your choice of operations : 5


PROGRAM EXITED !
```

Conclusion:

Thus we had Implemented Circular Queue using Array and performed following operations :

1) Insertion (Enqueue)
2) Deletion (Dequeue)
3) Display