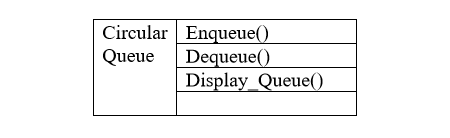
**PRACTICAL – 5**

**Aim: Implement basic operations of circular queue using an array.**

**Input: Size of the data structure, circular queue operation.**

**Output: Circular queue contents after every operation Algorithm**

***Algorithm:-***

**Q – Is data structure to hold elements**

**X – Is the data element which will be stored in Q**

**R – Is used to get last item from queue Q**

**F – Is used to get Front item from Q**

**N – Is the size of Q data structure**

**(Before Insertion of any element i.e. when Q is empty R=F= -1)**

1. **Enqueue(Q,R,F,X,N)**
2. **If ( R = N-1) then**

**Write (overflow) goto step iv**

**Else**

**R <- R+1**

1. **Q[R] <- X**
2. **If (F = -1) then**

**F<=0**

1. **END**

***Explanation:-***

According to algorithm

1. When R=N-1 i.e. R=size of Q no new element can be insert so print overflow and end the function
2. If not the case **1** then increment R with one where R points to the next index of last element of Q
3. Store new element X at index R in Q.
4. Checking if it is first element to be inserted i.e. if F=-1 then assign F=0. Here F will be used in deletion operation to findout whether Q is full or empty.
5. **Dequeue (Q,F,R)**
6. **Check for queue underflow**

**If(F = -1) then**

**Write “Queue underflow”**

**End**

1. **Delete element**

**Y <- Q[F]**

1. **Set front**

**If(F=R) then**

**F <- R <- -1**

**Else**

**F <- F+1**

1. **Return element**

**Return Y**

***Explanation:-***

1. Check if queue is empty i.e. if F=-1 then print underflow and end function.
2. If Q is not empty then Deleting the element pointed by index F and storing deleted element in Y. Here F points to front element of Queue Q.
3. Now checking if front end and last end index are same then delete operation further cannot be achive so resetting R and F to -1 indicating now Q is empty.

If not the above case then increment F with 1 to update next available data as front end of queue.

1. Returning deleted element stored in Y.

***Code:-***

**#include<iostream>**

**using namespace std;**

**int N=5,R=-1,F=-1;**

**void CQInsert(int X,int Q[]){**

**if((R==(N-1) && F==0)||(F==R+1)){**

**cout<<"OverFlow"<<endl;**

**} else {**

**if(R==N-1){**

**R=0;**

**} else {**

**R=R+1;**

**}**

**Q[R]=X;**

**if(F==-1){**

**F=0;**

**}**

**}**

**}**

**int CQDelete(int Q[]){**

**int y;**

**if(F==-1){**

**cout<<"Queue underflow"<<endl;**

**return -10;**

**} else {**

**y=Q[F];**

**if(F==R){**

**F=-1;**

**R=-1;**

**}**

**else if(F==N-1){**

**F=0;**

**} else {**

**F=F+1;**

**}**

**return y;**

**}**

**}**

**void display\_queue(int Q[]){**

**if(R>=F){**

**cout<<"Queue elements are :";**

**for(int i=F;i<=R;i++){**

**cout<<Q[i]<<" ";**

**}**

**cout<<endl;**

**} else {**

**cout<<"Queue elements are :";**

**for(int i=0;i<=R;i++){**

**cout<<Q[i]<<" ";**

**}**

**for(int i=R;i<F-1;i++){**

**cout<<"E ";**

**}**

**for(int i=F;i<N;i++){**

**cout<<Q[i]<<" ";**

**}**

**cout<<endl;**

**}**

**}**

**int main(){**

**int ch,val;**

**cout<<"Enter size of circular queue : "; cin>>N;**

**int Q[N];**

**do{**

**cout<<"1. Insert in queue."<<endl;**

**cout<<"2. delete from queue."<<endl;**

**cout<<"3. Display Queue"<<endl;**

**cout<<"4. to EXIT"<<endl;**

**cout<<"Enter choice : ";cin>>ch;**

**switch(ch){**

**case 1:**

**cout<<"Enter value in queue :";cin>>val;**

**CQInsert(val,Q);**

**break;**

**case 2:**

**CQDelete(Q);**

**break;**

**case 3:**

**display\_queue(Q);**

**break;**

**case 4:**

**cout<<"terminated..."<<endl;**

**break;**

**default:**

**cout<<"Wrong input."<<endl;**

**break;**

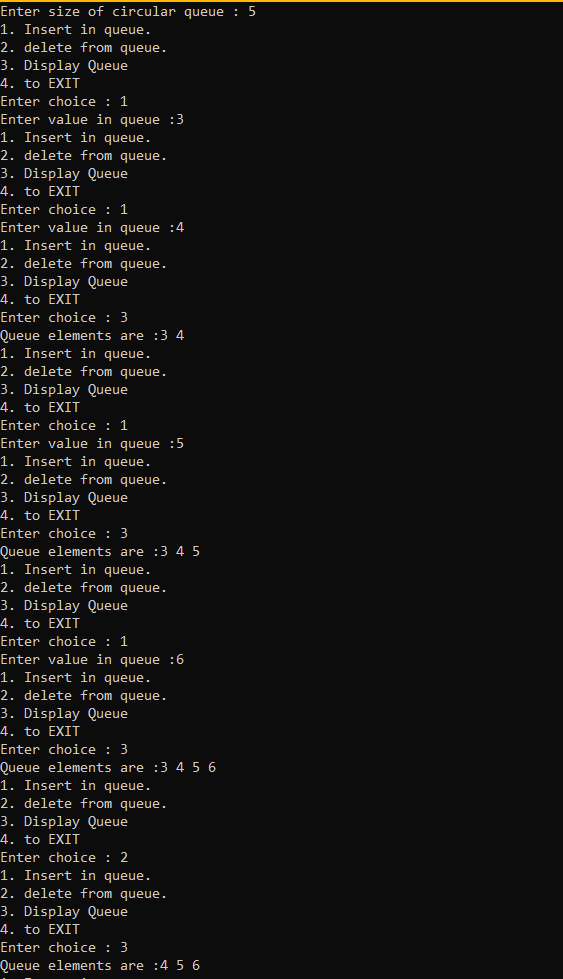
**}**

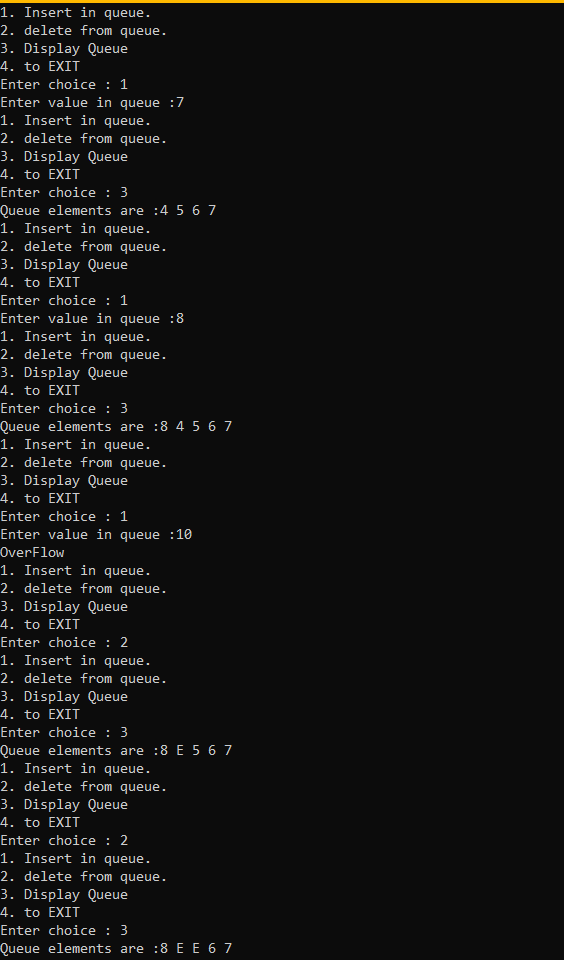
**}while(ch!=4);**

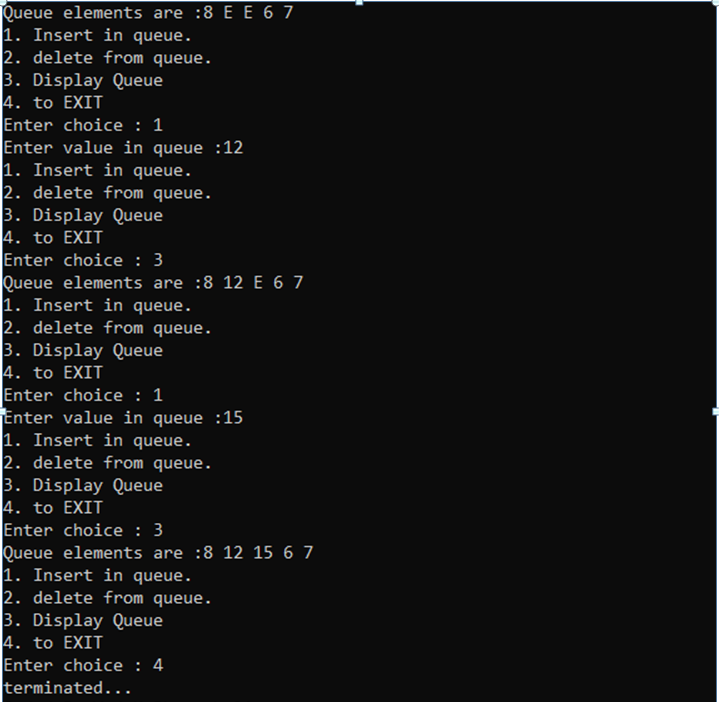
**return 0;**

**}**

**OUTPUT :**







**Conclusion :**

We learn the concept of reusing the space after deletion of elements in circular queue and In circular queue insertion and deletion operation follows first in first out order.