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

#define QueueLimit 34

typedef int QueueElementType;

typedef struct {
    int Front, Rear;
    QueueElementType Element[QueueLimit];
} QueueType;

typedef enum {FALSE, TRUE} boolean;

void CreateQ(QueueType *Queue);
void RemoveQ(QueueType *Queue, QueueElementType *Item);
void AddQ(QueueType *Queue, QueueElementType Item);
void TraverseQ(QueueType Queue);
boolean EmptyQ(QueueType Queue);
boolean FullQ(QueueType Queue);
boolean SearchQ(QueueType *Queue, QueueElementType Item);

int main()
{
    QueueType myQueue;
    QueueElementType i,searchItem;

    CreateQ(&myQueue);

    /*????????? ??? ???? ?? ?? ??????????? ??? 3.*/
    for(i = 3; i < 100; i+=3)
        AddQ(&myQueue,i);

    /*????????? ??? ????.*/
    TraverseQ(myQueue);

    /*?????? ?? ???????? ???? ?????????*/
    printf("Give the search value:");
    scanf("%d",&searchItem);

    /*??????????? ??? ????????? SearchQ ?? ???????
    ??? ??????? ??? ???? ??? ?? ????????? ??????
    ?????????? ??????.*/
    if(SearchQ(&myQueue,searchItem))
    {
        printf("Found\n");
        TraverseQ(myQueue);
    }
    else
     printf("Item not Found");

    return 0;
}

void CreateQ(QueueType *Queue)
{
    Queue->Front = 0;
    Queue->Rear = 0;
}

void RemoveQ(QueueType *Queue, QueueElementType *Item)
{
    if(!EmptyQ(*Queue))
    {
        *Item = Queue ->Element[Queue -> Front];
        Queue ->Front  = (Queue ->Front + 1) % QueueLimit;
    }
    else
        printf("Empty Queue");
}

void AddQ(QueueType *Queue, QueueElementType Item)
{
    int NewRear;

    if(!FullQ(*Queue))
    {
        NewRear = (Queue ->Rear + 1) % QueueLimit;
        Queue ->Element[Queue ->Rear] = Item;
        Queue ->Rear = NewRear;
    }
    else
        printf("Full Queue");
}

void TraverseQ(QueueType Queue) {
    int current;
    current = Queue.Front;
    while (current != Queue.Rear) {
        printf("%d ", Queue.Element[current]);
        current = (current + 1) % QueueLimit;
    }
    printf("\n");
}

boolean EmptyQ(QueueType Queue)
{
    return (Queue.Front == Queue.Rear);
}

boolean FullQ(QueueType Queue)
{
    return ((Queue.Front) == ((Queue.Rear +1) % QueueLimit));
}

boolean SearchQ(QueueType *Queue, QueueElementType Item)
{
    /*? ????????? found ????? ????? flag ,?? ??????? ? ???? ???
    ??????? ??? ??? ????????? ?????? ????????? ?? ????.*/
    int current,found=0;
    current = (*Queue).Front;

    /*???????? ??? ?? ???????? ??? ????? ??? ???? ??? ,?????
    ??? ????? ???? ?? ?? ????????? ??? ???????? ?? ???????????.
    ???? ?????? ??? ?? ?????? ???? ??? ???????? ? ???? ??? ??????????
    found ??????? 1 ,?????????? ??? ????????? ??? ?????????? ???
    ?????? ????????? ??? ???? ??? found.*/
    while(current != (*Queue).Rear)
    {
        if(Queue ->Element[current] != Item)
            RemoveQ(&(*Queue),&Queue ->Element[current]);
        else
        {
            RemoveQ(&(*Queue),&Queue ->Element[current]);
            found=1;
            break;
        }

        current = (current + 1) % QueueLimit;
    }

    return found;
```

```

}

```