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

#define QueueLimit 20

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);
void Move(QueueType *Queue,int number);

int main()
{
    QueueType EvenQueue,OddQueue;
    int i,x,numberOfEvenItems,numberOfOddItems;

    /*????????? ????????? ??????? ???????.*/
    srand(time(NULL));

    CreateQ(&EvenQueue);
    CreateQ(&OddQueue);

    /*????????? ???? ???????? ??? ?? ????? ?????? ? ????????
    ??? ??????? ???? ??????? ????.*/
    for(i=0; i < 20;i++)
    {
        x=rand()%20;

        if(x%2)
            AddQ(&OddQueue,x);
        else
            AddQ(&EvenQueue,x);

    }

    /*???????? ??? ???????? ??? ????? ??? ??? ???????????? ???.*/
    printf("Size of EvenQueue:%d\n",EvenQueue.Rear);
    TraverseQ(EvenQueue);

    /*???????? ??? ???????? ??? ????? ??? ??? ???????????? ???.*/
    printf("Size of OddQueue:%d\n",OddQueue.Rear);
    TraverseQ(OddQueue);

    /*????????? ?????? ?????? ??? ?? 1 ??? ?? ??????? ??? ????? ???
    ?? ??? ????????? move ?????????? ??? ???????????????? ?? ??????
    ???? ??? ????? ??? ?????.*/
    numberOfEvenItems=1+rand()%EvenQueue.Rear;
    printf("Random number of items=%d\n",numberOfEvenItems);
    Move(&EvenQueue,numberOfEvenItems);
    TraverseQ(EvenQueue);

    /*????????? ?????? ?????? ??? ?? 1 ??? ?? ??????? ??? ????? ???
    ?? ??? ????????? move ?????????? ??? ???????????????? ?? ??????
    ???? ??? ????? ??? ?????.*/
```

```
    numberOfOddItems=1+rand()%OddQueue.Rear;
    printf("Random number of items=%d\n",numberOfOddItems);
    Move(&OddQueue,numberOfOddItems);
    TraverseQ(OddQueue);

    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));
}

/*????? ?????????? ????? ??? ?? ?????????? ?? ?????? ??? ???????
???? ???? ?????????? ??? ???? ???????? ?? ???? ??????????????????
??? ????? ??? ??????? ?????.*/
void Move(QueueType *Queue,int number)
{
    QueueType TempQueue;
    int i;
```

```
133
134     CreateQ(&TempQueue);
135
136     for(i=0;i<number;i++)
137     {
138         AddQ(&TempQueue,Queue ->Element[Queue ->Front]);
139         RemoveQ(&(*Queue),&Queue ->Element[Queue ->Front]);
140     }
141
142     for(i=0;i<number;i++)
143     {
144         AddQ(&(*Queue),TempQueue.Element[TempQueue.Front]);
145         RemoveQ(&TempQueue,&TempQueue.Element[TempQueue.Front]);
146     }
147 }
```