UNIT-II
Queues

Queue:

Queue is an ordered collection of items from which items may be deleted at one end (called the **front** of the Queue) and into which items may be inserted at the other end (called **rear** of the Queue).

The first inserted element into the Queue is the first deleted element. For this reason, a Queue is sometimes called First-In-First-Out (FIFO).

Array representation of Queue:

Use an array to hold the elements of the Queue and to use two variables, front and rear to hold the positions within the array of the first and last elements of the Queue.

```
#define max 10
struct queue
{
        int items[max];
        int front, rear;
};
struct queue q;
```

Initially **q.front = 0; and q.rear = -1;**

The queue is empty whenever q.front > q.rear.

No of elements in Queue = q.rear – q.front + 1;

Operations on Queue:

Empty operation;

```
int empty(struct queue *q)
{
        if(q→rear < q→front)
                return 1;
        else
                return 0;
}
```

Insert operation:

Using an array to hold a queue, check overflow condition.

```
void insert(struct queue *q, int a)
{
        if(q→rear = = max-1)
                printf("\nQueue is overflow");
        else
```

```
        {
                q→rear++;
                q→items[q→rear] = a;
        }
}
```

Remove operation:

Check empty condition.

```
int remove(struct queue *q)
{
        int a;
        if(q→rear < q→front)
                return -1;
        else
        {
                a = q→items[q→front];
                q→front++;
                return a;
        }
}
```

Peek operation:

```
int peek(struct queue *qu)
{
        if(qu->front>qu->rear)
                return -1;
        else
                return(qu->items[qu->front]);
}
```

/*queue operations using Array program*/

```
/*queue operations using Array*/
#define max 10
#include<stdio.h>
struct queue
{
        int front,rear;
        int items[max];
};
void insert(struct queue *,int);
int del(struct queue *);
int empty(struct queue *);
int peek(struct queue *);
```

```
void display(struct queue *);
main()
{
        int choice,x;
        struct queue q;
        q.front=0;
        q.rear=-1;
        while(1)
        {
        printf("\n\n**************************\n\n");
        printf("\t\tMENU\n");
        printf("\n**************************\n");
        printf("1.insert\n2.delete\n3.empty\n4.peek\n5.display\n6.exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:printf("\nEnter the element:");
                        scanf("%d",&x);
                        insert(&q,x);
                        printf("\nElements are:");
                        display(&q);
                        break;
                case 2: x=del(&q);
                        if(x==-1)
                                printf("\nqueue is empty");
                        else
                                printf("\ndeleted element is %d",x);
                        printf("\nelements are:");
                        display(&q);
                        break;
                case 3: x=empty(&q);
                        if(x==1)
                                printf("\nqueue is empty");
                        else
                                printf("\nqueue is not empty");
                        break;
                case 4: x=peek(&q);
                        if(x==-1)
                                printf("\nQueue is empty");
                        else
                                printf("\nFront element is:%d",x);
                        break;
```

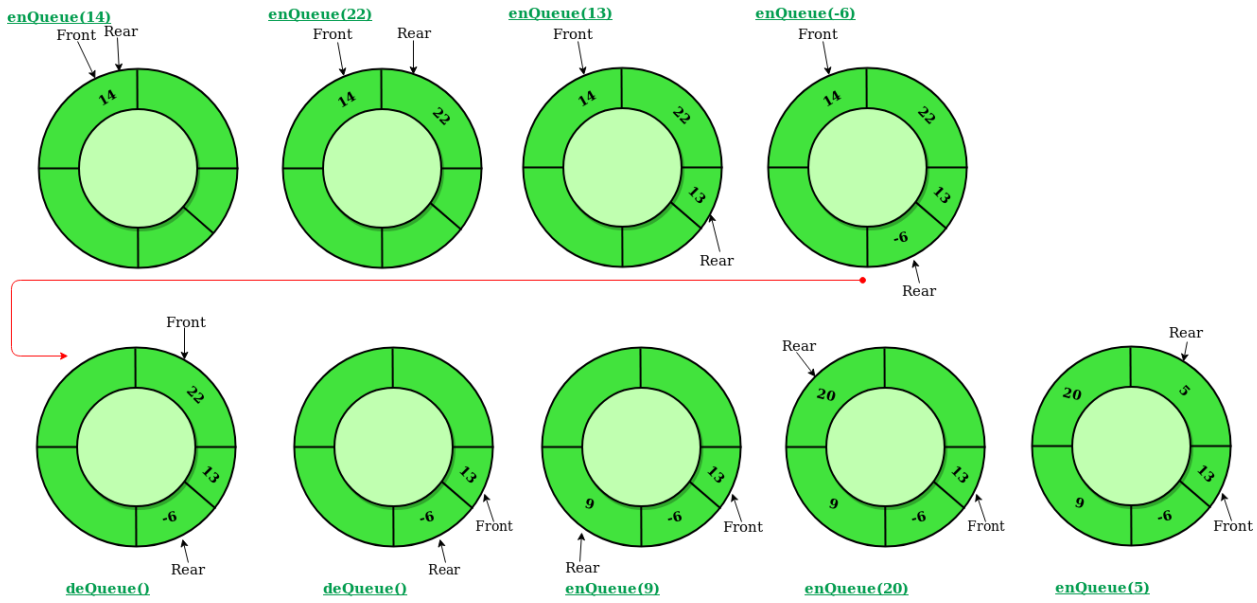
```
                case 5: printf("\nElements are\n");
                        display(&q);
                        break;
                case 6: exit(0);
        }
        }
}
void insert(struct queue *qu,int a)
{
        if(qu->rear==max-1)
        {
                printf("\nqueue is full");

        }
        else
        {
                qu->rear++;
                qu->items[qu->rear]=a;
        }
}
int del(struct queue *qu)
{
        int a;
        if(empty(qu))
                return -1;
        else
        {
                a=qu->items[qu->front];
                qu->front++;
                return a;
        }
}
void display(struct queue *qu)
{
        int i;
        for(i=(qu->front);i<=qu->rear;i++)
                printf("%d\t",qu->items[i]);
}
int empty(struct  queue *qu)
{
        if(qu->front>qu->rear)
                return 1;
        else
```

```
                return 0;
}
int peek(struct queue *qu)
{
        if(qu->front>qu->rear)
                return -1;
        else
                return qu->items[qu->front];
}
```

Array representation of Circular Queue:

The main disadvantage of linear queue using array is that when elements are deleted from the queue, new elements cannot be added in their place in the queue, i.e. the position cannot be reused.



Use an array to hold the elements of the Circular Queue and to use two variables, front and rear to hold the positions within the array of the first and last elements of the Circular Queue.

```
#define max 10
struct cqueue
{
        int items[max];
        int front, rear;
};
struct cqueue cq;
```

Initially **q.front = -1; and q.rear = -1;**

Operations on Circular Queue:

Empty operation.

```
int empty(struct  cqueue *cqu)
{
        if(cqu->front= =-1)
                return 1;
        else
                return 0;
}
```

Insert operation:

Using an array to hold a Circular queue, check overflow condition.

```
void insert(struct cqueue *cqu,int a)
{
        if(((cqu->front==0)&&(cqu->rear==max-1))||(cqu->front==cqu->rear+1))
        {
                printf("\nCircular queue is Overflow");
                return;
        }
        else if(cqu->front==-1)
        {
                cqu->front=0;
                cqu->rear=0;
        }
        else
        {
                if(cqu->rear==max-1)
                        cqu->rear=0;
                else
                        cqu->rear=cqu->rear+1;
        }
        cqu->items[cqu->rear]=a;
}
```

Remove operation:

Check empty condition.

```
int del(struct cqueue *cqu)
{
        int a;
        if(cqu->front==-1)
                return -1;
```

```
        a=cqu->items[cqu->front];
        if(cqu->front==cqu->rear)
        {
                cqu->front=-1;
                cqu->rear=-1;
        }
        else
        {
                if(cqu->front==max-1)
                        cqu->front=0;
                else
                        cqu->front++;
        }
        return a;
}
```

Display:

```
void display(struct cqueue *cqu)
{
        int i;
        if(cqu->front==-1)
                printf("Circular Queue is Empty");
        else
        {
                if(cqu->front<=cqu->rear)
                        for(i=cqu->front;i<=cqu->rear;i++)
                                printf("%d\t",cqu->items[i]);
                else
                {
                        for(i=cqu->front;i<=max-1;i++)
                                printf("%d\t",cqu->items[i]);
                        for(i=0;i<=cqu->rear;i++)
                                printf("%d\t",cqu->items[i]);
                }
        }
}
```

Peek operation:

```
int peek(struct  cqueue *cqu)
{
        int b;
        if(cqu->front==-1)
                return -1;
        else
```

```
                return cqu->items[cqu->front];
}
```

/* CIRCULAR QUEUE OPERATIONS PROGRAM */

```
#define max 10
#include<stdio.h>
struct cqueue
{
        int front,rear;
        int items[max];
};
void insert(struct queue *,int);
int del(struct queue *);
int empty(struct queue *);
int peek(struct queue *);
void display(struct queue *);
main()
{
        int choice,x;
        struct cqueue cq;
        cq.front=cq.rear=-1;
        while(1)
        {
        printf("\n\n*****************************\n\n");
        printf("\t\tMENU\n");
        printf("\n*****************************\n");
        printf("1.insert\n2.delete\n3.peek\n4.Empty\n5.Display\n6.exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:printf("\nEnter the element:");
                        scanf("%d",&x);
                        insert(&cq,x);
                        printf("\nElements are:");
                        display(&cq);
                        break;
                case 2: x=del(&cq);
                        if(x==-1)
                                printf("\nqueue is empty");
                        else
                                printf("\ndeleted element is %d",x);
```

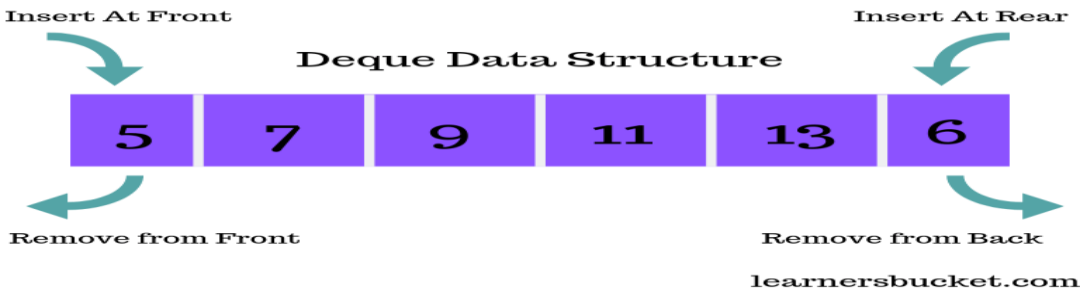
```
                printf("\nelements are:");
                display(&cq);
                break;
        case 3: x=peek(&cq);
                if(x==-1)
                        printf("\nCircular queue is empty");
                else
                        printf("\nfirst element in Circular queue is:%d",x);
                break;
        case 4: x=empty(&cq);
                if(x==1)
                        printf("\nqueue is empty");
                else
                        printf("\nqueue is not empty");
                break;
        case 5: printf("\nElements are\n");
                display(&cq);
                break;
        case 6: exit(0);
    }
    }
}
void insert(struct cqueue *cqu,int a)
{
    if(((cqu->front==0)&&(cqu->rear==max-1))||(cqu->front==cqu->rear+1))
    {
        printf("\nCircular queue is Overflow");
        return;
    }
    else if(cqu->front==-1)
    {
        cqu->front=0;
        cqu->rear=0;
    }
    else
    {
        if(cqu->rear==max-1)
            cqu->rear=0;
        else
            cqu->rear=cqu->rear+1;
    }
    cqu->items[cqu->rear]=a;
}
```

```
int del(struct cqueue *cqu)
{
	int a;
	if(cqu->front==-1)
		return -1;
	a=cqu->items[cqu->front];
	if(cqu->front==cqu->rear)
	{
		cqu->front=-1;
		cqu->rear=-1;
	}
	else
	{
		if(cqu->front==max-1)
			cqu->front=0;
		else
			cqu->front++;
	}
	return a;
}
void display(struct cqueue *cqu)
{
	int i;
	if(cqu->front==-1)
		printf("Circular Queue is Empty");
	else
	{
		if(cqu->front<=cqu->rear)
			for(i=cqu->front;i<=cqu->rear;i++)
				printf("%d\t",cqu->items[i]);
		else
		{
			for(i=cqu->front;i<=max-1;i++)
				printf("%d\t",cqu->items[i]);
			for(i=0;i<=cqu->rear;i++)
				printf("%d\t",cqu->items[i]);
		}
	}
}
```

```
int empty(struct  cqueue *cqu)
{
        int b;
        if(cqu->front==-1)
                return 1;
        else
                return 0;
}
int peek(struct  cqueue *cqu)
{
        int b;
        if(cqu->front==-1)
                return -1;
        else
                return cqu->items[cqu->front];
}
```

Dequeues:

Deque or Double Ended Queue is a type of queue in which insertion and removal of elements can either be performed from the front or the rear. Thus, it does not follow FIFO rule (First In First Out).



Types of Deque

- **Input Restricted Deque**
  In this deque, input is restricted at a single end but allows deletion at both the ends.
- **Output Restricted Deque**
  In this deque, output is restricted at a single end but allows insertion at both the ends.

Array representation of Deque:

```
#define max 10
struct deque
{
        int items[max];
        int front, rear;
};
struct deque dq;
```

Initially dq.front = -1; and dq.rear = -1;

Operations on deque:

Empty operation.

```
int empty(struct  deque *dq)
{
        if(dq->front==-1)
                return 1;
        else
                return 0;
}
```

Insert operation:

Using an array to hold a Circular queue, check overflow condition.

```
void insert_front(struct deque *dq,int a)
{
        if(((dq->front==0)&&(dq->rear==max-1))||(dq->front==dq->rear+1))
        {
                printf("\nCircular queue is Overflow");
                return;
        }
        else if(dq->front==-1)
        {
                dq->front=0;
                dq->rear=0;
        }
        else
        {
                if(dq->front==0)
                        dq->front=max-1;
                else
                        dq->front=dq->front-1;
        }
```

```
        dq->items[dq->front]=a;
}
void insert_rear(struct deque *dq,int a)
{
        if(((dq->front==0)&&(dq->rear==max-1))||(dq->front==dq->rear+1))
        {
                printf("\nCircular queue is Overflow");
                return;
        }
        else if(dq->front==-1)
        {
                dq->front=0;
                dq->rear=0;
        }
        else
        {
                if(dq->rear==max-1)
                        dq->rear=0;
                else
                        dq->rear=dq->rear+1;
        }
        dq->items[dq->rear]=a;
}
```

Remove operation:

Check empty condition.

```
int del_front(struct deque *dq)
{
        int a;
        if(dq->front==-1)
                return -1;
        a=dq->items[dq->front];
        if(dq->front==dq->rear)
        {
                dq->front=-1;
                dq->rear=-1;
        }
        else
        {
                if(dq->front==max-1)
                        dq->front=0;
                else
                        dq->front++;
```

```
        }
        return a;
}
int del_rear(struct deque *dq)
{
        int a;
        if(dq->front==-1)
                return -1;
        a=dq->items[dq->rear];
        if(dq->front==dq->rear)
        {
                dq->front=-1;
                dq->rear=-1;
        }
        else
        {
                if(dq->rear==0)
                        dq->rear=max-1;
                else
                        dq->rear--;
        }
        return a;
}
```

Display:

```
void display(struct deque *dq)
{
        int i,j;
        i=dq->front;
        j=dq->rear;
        if(dq->front==-1)
                printf("Circular Queue is Empty");
        else
        {
                if(i<=j)
                        while(i<=j)
                        {
                                printf("%d\t",dq->items[i]);
                                i++;
                        }
                else
                {
                        while(i<=max-1)
                        {
```

```
                        printf("%d\t",dq->items[i]);
                        i++;
                }
                i=0;
                while(i<=j)
                {
                        printf("%d\t",dq->items[i]);
                        i++;
                }
        }
    }

}
```

Peek Operation:

```
int peek_front(struct  deque *dq)
{
    if(dq->front==-1)
        return -1;
    else
        return dq->items[dq->front];
}
int peek_rear(struct deque *dq)
{
    if(dq->front==-1)
        return -1;
    else
        return dq->items[dq->rear];
}
```

/* DOUBLE ENDED QUEUE OPERATIONS PROGRAM*/

```
#define max 10
#include<stdio.h>
struct deque
{
    int front,rear;
    int items[max];
};
void insert_front(struct deque *,int);
void insert_rear(struct deque *,int);
int del_front(struct deque *);
int del_rear(struct deque *);
int empty(struct deque *);
int peek_front(struct deque *);
```

```
int peek_rear(struct deque *);
void display(struct deque *);
main()
{
        int choice,x;
        struct deque dq;
        dq.front=dq.rear=-1;
        while(1)
        {
        printf("\n\n*************************\n\n");
        printf("\t\tMENU\n");
        printf("\n*************************\n");
        printf("1.insert at front\n2.insert at rear\n3.delete at front\n4.delete at rear");
        printf("\n5.peek at front\n6.peek at rear\n7.Empty\n8.Display\n9.exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:printf("\nEnter the element:");
                        scanf("%d",&x);
                        insert_front(&dq,x);
                        printf("\nElements are:");
                        display(&dq);
                        break;
                case 2: printf("\nEnter the element:");
                        scanf("%d",&x);
                        insert_rear(&dq,x);
                        printf("\nElements are:");
                        display(&dq);
                        break;
                case 3: x=del_front(&dq);
                        if(x==-1)
                                printf("\nqueue is empty");
                        else
                                printf("\ndeleted element is %d",x);
                        printf("\nelements are:");
                        display(&dq);
                        break;
                case 4: x=del_rear(&dq);
                        if(x==-1)
                                printf("\nqueue is empty");
                        else
                                printf("\ndeleted element is %d",x);
```

```
                printf("\nelements are:");
                display(&dq);
                break;
        case 5: x=peek_front(&dq);
                if(x==-1)
                        printf("\nCircular queue is empty");
                else
                        printf("\nfirst element in Circular queue is:%d",x);
                break;
        case 6: x=peek_rear(&dq);
                if(x==-1)
                        printf("\nCircular queue is empty");
                else
                        printf("\nlast element in Circular queue is:%d",x);
                break;
        case 7: x=empty(&dq);
                if(x==1)
                        printf("\nqueue is empty");
                else
                        printf("\nqueue is not empty");
                break;
        case 8: printf("\nElements are\n");
                display(&dq);
                break;
        case 9: exit(0);
    }
    }
}
void insert_front(struct deque *dq,int a)
{
    if(((dq->front==0)&&(dq->rear==max-1))||(dq->front==dq->rear+1))
    {
        printf("\nCircular queue is Overflow");
        return;
    }
    else if(dq->front==-1)
    {
        dq->front=0;
        dq->rear=0;
    }
    else
    {
        if(dq->front==0)
```

```
                dq->front=max-1;
        else
                dq->front=dq->front-1;
    }
    dq->items[dq->front]=a;
}
void insert_rear(struct deque *dq,int a)
{
    if(((dq->front==0)&&(dq->rear==max-1))||(dq->front==dq->rear+1))
    {
        printf("\nCircular queue is Overflow");
        return;
    }
    else if(dq->front==-1)
    {
        dq->front=0;
        dq->rear=0;
    }
    else
    {
        if(dq->rear==max-1)
            dq->rear=0;
        else
            dq->rear=dq->rear+1;
    }
    dq->items[dq->rear]=a;
}
int del_front(struct deque *dq)
{
    int a;
    if(dq->front==-1)
        return -1;
    a=dq->items[dq->front];
    if(dq->front==dq->rear)
    {
        dq->front=-1;
        dq->rear=-1;
    }
    else
    {
        if(dq->front==max-1)
            dq->front=0;
        else
```

```
                dq->front++;
        }
        return a;
}
int del_rear(struct deque *dq)
{
        int a;
        if(dq->front==-1)
                return -1;
        a=dq->items[dq->rear];
        if(dq->front==dq->rear)
        {
                dq->front=-1;
                dq->rear=-1;
        }
        else
        {
                if(dq->rear==0)
                        dq->rear=max-1;
                else
                        dq->rear--;
        }
        return a;
}
void display(struct deque *dq)
{
        int i,j;
        i=dq->front;
        j=dq->rear;
        if(dq->front==-1)
                printf("Circular Queue is Empty");
        else
        {
                if(i<=j)
                        while(i<=j)
                        {
                                printf("%d\t",dq->items[i]);
                                i++;
                        }
                else
                {
                        while(i<=max-1)
                        {
```

```
                        printf("%d\t",dq->items[i]);
                        i++;
                }
                i=0;
                while(i<=j)
                {
                        printf("%d\t",dq->items[i]);
                        i++;
                }
        }
    }

}
int empty(struct  deque *dq)
{
    if(dq->front==-1)
        return 1;
    else
        return 0;
}
int peek_front(struct  deque *dq)
{
    if(dq->front==-1)
        return -1;
    else
        return dq->items[dq->front];
}
int peek_rear(struct deque *dq)
{
    if(dq->front==-1)
        return -1;
    else
        return dq->items[dq->rear];
}
```

Priority Queues:

Priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order of in which the elements will be processed.

The general rules for processing the elements of the priority queue are

1. An element with higher priority is processed before an element with a lower priority.
2. Two elements have the same priority are processed on a first come first serve (FCFS) basis.

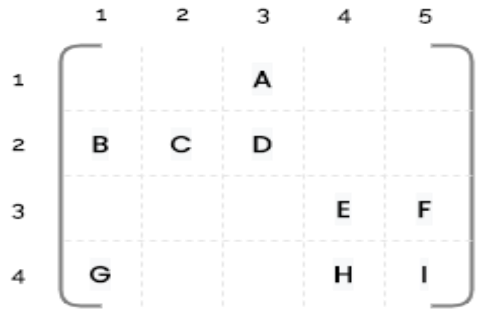
When an element to be removed from the queue, the element with the highest priority is retrieved first.

Implementation of priority queue:
There are two ways to implement a priority queue

1. We store the elements in unsorted order. So that insertion is always done at the end of the list. Every time when an element to be removed from the list, the element with highest priority will be searched and removed.
2. We store the elements in sorting order. So when an element has to be removed, the queue will not have to be searched for the element with the highest priority.

Practically both these techniques are inefficient.

Array representation of a priority queue:
When arrays are used to implement a priority queue, then a separate queue for each priority number is maintained. Each of these queues will be implemented using circular queues. Every individual queue will have its own front and rear pointers. We use a two-dimensional array where each queue will be allocated the same amount of space.



FRONT	REAR
3	3
1	3
4	5
4	1

Insertion:
To insert a new element with priority k in the priority queue, add the element at the rear end of row k, where k is the row number as well as the priority number of that element.
Deletion:
To delete an element, we find the first non-empty queue and then process the front element.

Multiple Queues:
An array is used to represent two queues queue1 and queue2. The value of n in such that the combined size of both queues will never exceed n. while operating on these queues queue1 will grow from left to right and queue2 will grow from right to left.

/*Multiple QUEUE operations Program*/

```
#define max 10
#include<stdio.h>
struct queue
```

```
{
        int front1,rear1,front2,rear2;
        int items[max];
};
void insert1(struct queue *,int);
void insert2(struct queue *,int);
int del1(struct queue *);
int del2(struct queue *);
int empty1(struct queue *);
int empty2(struct queue *);
void display1(struct queue *);
void display2(struct queue *);
int peek1(struct queue *);
int peek2(struct queue *);
main()
{
        int choice,x,ch;
        struct queue q;
        q.front1=-1;
        q.rear1=-1;
        q.front2=max;
        q.rear2=max;
        while(1)
        {
        printf("\n\n*****************************\n\n");
        printf("\t\tMENU\n");
        printf("\n*****************************\n");
        printf("1.insert\n2.delete\n3.display\n4.empty\n5.peek\n6.exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:printf("\nEnter the element:");
                        scanf("%d",&x);
                        printf("insert in\n1.queue-1\n2.queue-2\n");
                        printf("Enter your choice:");
                        scanf("%d",&ch);
                        if(ch==1)
                                insert1(&q,x);
                        else if(ch==2)
                                insert2(&q,x);
                        else
                                printf("\nwrong choice");
```

```
            break;
case 2: printf("delete in\n1.queue-1\n2.queue-2\n");
            printf("Enter your choice:");
            scanf("%d",&ch);
            if(ch==1)
                  x=del1(&q);
            else if(ch==2)
                  x=del2(&q);
            else
                  printf("\nwrong choice");
            if(x==-1)
                  printf("\nqueue is underflow");
            else
                  printf("\ndeleted element is %d",x);
            break;
case 4: printf("Check empty in\n1.queue-1\n2.queue-2\n");
            printf("Enter your choice:");
            scanf("%d",&ch);
            if(ch==1)
                  x=empty1(&q);
            else if(ch==2)
                  x=empty2(&q);
            else
                  printf("\nwrong choice");

            if(x==1)
                  printf("\nqueue is empty");
            else
                  printf("\nqueue is not empty");
            break;
case 3: printf("Display in\n1.queue-1\n2.queue-2\n");
            printf("Enter your choice:");
            scanf("%d",&ch);
            printf("\nElements are\n");
            if(ch==1)
                  display1(&q);
            else if(ch==2)
                  display2(&q);
            else
                  printf("\nwrong choice");
            break;
case 5: printf("Peek in\n1.queue-1\n2.queue-2\n");
            printf("Enter your choice:");
```

```
                        scanf("%d",&ch);
                        if(ch==1)
                                x=peek1(&q);
                        else if(ch==2)
                                x=peek2(&q);
                        else
                                printf("\nwrong choice");
                        if(x==-1)
                                printf("\nqueue is empty");
                        else
                                printf("\nfront element of queue:%d",x);
                        break;
                case 6: exit(0);
        }
        }
}
void insert1(struct queue *q,int a)
{
        if(q->rear1==q->rear2-1)
        {
                printf("\nqueue is overflow");

        }
        else
        {
                if(q->rear1==-1 && q->front1==-1)
                {
                        q->front1=q->rear1=0;
                        q->items[q->rear1]=a;
                }
                else
                {
                        q->rear1++;
                        q->items[q->rear1]=a;
                }
        }
}
void insert2(struct queue *q,int a)
{
        if(q->rear1==q->rear2-1)
        {
                printf("\nqueue is overflow");
```

```
        }
        else
        {
                if(q->rear2==max && q->front2==max)
                {
                        q->front2=q->rear2=max-1;
                        q->items[q->rear2]=a;
                }
                else
                {
                        q->rear2--;
                        q->items[q->rear2]=a;
                }
        }
}
int del1(struct queue *q)
{
        int a;
        if(q->front1==-1)
                return -1;
        else
        {
                a=q->items[q->front1];
                q->front1++;
                if(q->front1>q->rear1)
                        q->rear1=q->front1=-1;
                return a;
        }

}
int del2(struct queue *q)
{
        int a;
        if(q->front2==max)
                return -1;
        else
        {
                a=q->items[q->front2];
                q->front2--;
                if(q->front2<q->rear2)
                        q->rear2=q->front2=max;
                return a;
        }
```

```
}
void display1(struct queue *q)
{
        int i;
        if(q->front1==-1)
                printf("\nQueue is empty");
        else
                for(i=q->front1;i<=q->rear1;i++)
                        printf("%d\t",q->items[i]);
}
void display2(struct queue *q)
{
        int i;
        if(q->front2==max)
                printf("\nQueue is empty");
        else
                for(i=q->front2;i>=q->rear2;i--)
                        printf("%d\t",q->items[i]);
}
int empty1(struct  queue *q)
{
        if(q->front1==-1)
                return 1;
        else
                return 0;
}
int empty2(struct  queue *q)
{
        if(q->front2==max)
                return 1;
        else
                return 0;
}
int peek1(struct queue *q)
{
        int a;
        if(q->front1==-1)
                return -1;
        else
        {
                a=q->items[q->front1];
                return a;
```

```
        }
}
int peek2(struct queue *q)
{
        int a;
        if(q->front2==max)
                return -1;
        else
        {
                a=q->items[q->front2];
                return a;
        }
}
```