QUEUES

Q.1 What is Queue

- It is a special kind of list, where items are inserted at one end (the rear) and deleted from the other end(front).
- Queue is FIFO (First In First Out) list.

- Eg:- 1) Queue at railway ticket counter.
  2) Queue for bus/rikshaw.
  3) Toll-tax bridge.

Q.2 Give ADT for Queue data structure, discuss two applications of queue data structure. [5M MAY17]

Application of queues:

- Various features of operating system are implemented using a queue.
- Scheduling of processes (round robin algorithm).
- Spooling(to maintain a queue of jobs to be printed).
- A queue of client processes waiting to receive from the server process.
- Queue is useful in CPU scheduling, Disk scheduling, when multiple process required CPU at same time.
- In call centre phone system will use queue to hold people calling them in an order, until a service representative is free.

Array representation and implementation of queues:

An array representation of queue requires three entities.

- An array to hold queue elements.
- A variable to hold the index of the front element.
- A variable to hold the index of the rear element.
- A queue data type may be defined formally as follows.

```
#define MAX 30
typedef struct Queue
{
        int data [MAX];
        int  front , rear;
}Queue;
```

If the queue is empty then front =-1 and rear=-1;
If the queue is full then rear= MAX-1;
If rear=front then queue contains just one element.
If rear > front then queue is not empty.

Operation on queue implemented

A set of operations on a queue includes:

- Initialize(): initializes a queue by setting the value of rear and front to – 1.
- Enqueue(): inserts an element at the rear end of the queue .
- Dequeue():delete the front element and returns the same.
- Empty(): it returns true (1)if the queue is empty and returns false(0) if the queue is not empty.
- Full(): it return true (1) if the queues is full and returns false (0) if the queue is not empty.
- Print(): printing of queue elements.

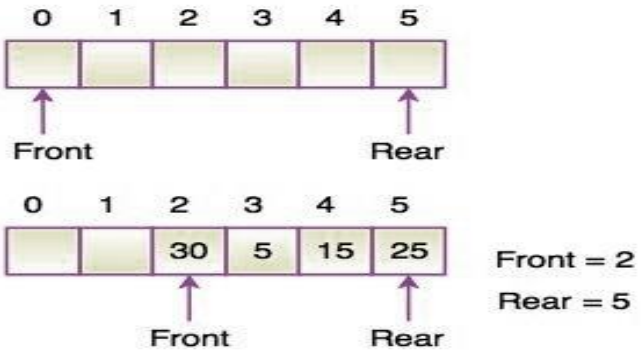
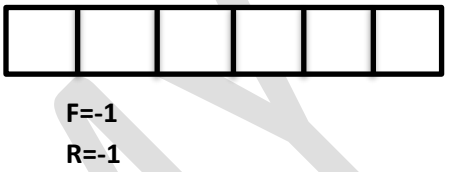


Fig. Implementation of Queue using Array

'C' function to initialize()

```
void initialize(Queue *p)
{
        p→R= - 1;
        p→F= -1;
}
```

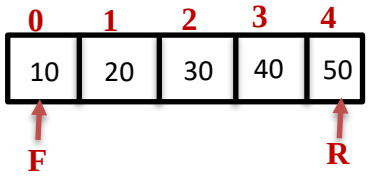


'C ' function to check whether queue is empty or not

```
int empty (Queue *P)
{
        if (p→R==-1)
                return (1);
        else
                return (0);
}
```

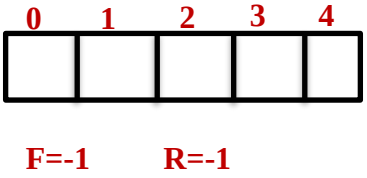
'C' function to check whether queue is full or not .

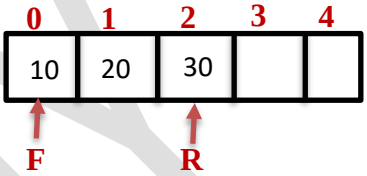
```
int full (Queue *p)
{
    if(p→R==MAX-1);
        return (1);
    else
        return(0);
}
```



'C' function for insertion in queue.

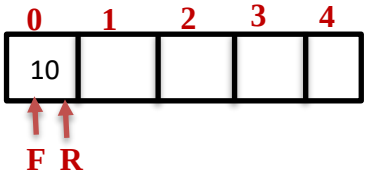
```
void enqueue (Queue *p, int x)
{
        if (p→R == -1)
        {
                p→R =p→F=0;
                p→data[p→R] = x;
        }
        else
        {
                p→R=p→R+1;
                p→data[p→R]=x;
        }
}
```

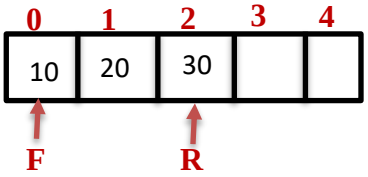




'C' function for deletion.

```
int dequeue (Q *p)
{
        int x;
        x=p→data[p→F];
        if (p→R==p→F)    /* last element */
        {
                p→R=-1;
                p→F=-1;
        }
        else
        {
                p→F = p→F +1 ;
        }
        return (x);
}
```





'C' function for printing a queue .

```
void print (Queue *p)
{
        for(i=p→F;i<=p→R;i++)
        {
                printf("\n %d", p→data[i]);
        }
}
```

Write a program to implement linear queue using array. [10M DEC17]

```
#include<stdio.h>
#include<conio.h>
#define MAX 30
typedef struct Queue
{
        int data[MAX] ;
        int F, R;
}Queue;
void initialize(Queue *p);
int empty (Queue *P);
int full (Queue   *p);
void enqueue (Queue *p, int x);
int dequeue (Queue *p);
void print (Queue * p) ;
void main()
{

        int x, ch, n;
        Queue q;
        init(&q);
        do
        {
                printf("1-create\n 2-insert\n 3-delete\n 4-Display 5-End\n")
                printf("Enter your choice n");
                scanf("%d",&ch);
```

```
switch(ch)
{
        case 1:
                printf("enter no of elements\n");
                scanf("%d",&n);
                init(&q);
                printf("enter the data\n");
                for(i=0;i<n;i++)
                {
                        scanf("%d",&x);
                        if (full(&q))
                        {
                                printf("queue is full\n");
                                Exit(0);
                        }
                        else
                        {
                                enqueue(&q,x);
                        }
                }
                 break;
        Case 2:
                printf("enter the element to be inserted");
                scanf("%d",&x);
                if (full(&q))
                {
                        printf("queue is full\n");
                        Exit(0);
                }
                else
                {
                        enqueue(&q,x);
                }
                break;
        Case 3:
```

```
                        if(empty(&q))
                        {
                                printf("queue is empty");
                                Exit(0);
                        }
                        else
                        {
                                x=dequeue(&q);
                                printf("element=%d",x);
                        }
                        break;
                Case 4:
                        display(&q);
                        break;
                default:
                }
    }while(ch!=5);
    getch();
}
void initialize(Queue *p)
{
    p→R=-1;
    p→F=-1;
}
int empty (Queue *P)
{
    if (p→R == -1)
            return(1);
    else
            return(0);
}
```

```
int full (Queue *p)
{
    if(p→R == MAX-1);
            return (1);
    else
            return(0);
}
void enqueue (Queue *p, int x)
{
    if (p→R == -1) /*empty queue  */
    {
            p→R =p→F=0;
            p→data[p->R] = x;
    }
    else
    {
            p→R=p→R+1;

    p→data[p→R]=x;
    }
}
int dequeue (Queue *p)
{
    int x;
    x=p→data[p→F];
    if (p→R = = p→F)    /* last element */
    {
            p→R=-1;
            p→F=-1;
    }
    else
    {
            p→F = p→F+1;
    }
    return (x);
```

```
 }
void print (Queue * p)
{
        int i;

        for(i=p→F;i<= p→R;i++)
        {
                printf("%d",p→data[i]);
        }
}
```

Write a menu driven program in C to implement QUEUE ADT the program should perform following operations. [12M MAY16]

- **i) Inserting element in the queue**
- **ii) Deleting an element from the queue.**
- **iii) Displaying the queue**
- **iv) Exiting the program**

```
#include<Stdio.h>
#include<conio.h>
#define MAX 30
typedef struct Q
{
      int data[MAX] ;
      int F, R;
}Q;
void initialize(Q *p)  ;
int empty (Q *P) ;
int full (Q   *p) ;
void enqueue (Q *p, int x);
int dequeue (Q *p);
void print (Q * p) ;
void main()
{

        int x, OP, n;
```

```
Q q;
printf("enter no of elements\n");
scanf("%d",&n);
init(&q);
printf("enter the data\n");
for(i=0;i<n;i++)
{
        scanf("%d",&x);
        if (full(&q))
        {
                printf("queue is full\n");
                exit(0);
        }
        else
        {
                enqueue(&q,x);
        }
}
do
{
        printf("1-Insert an element \n 2-Deleting an element\n 3-displaying
        the queue\n 4-Exit the program.\n")
        printf("Enter your choice n");
        scanf("%d",&op);
        switch(op)
        {
                case 1:
                        printf("enter the element to be inserted");
                        Scanf("%d",&x);
                        if (full(&q))
                        {
                                printf("queue is full\n");
                                Exit(0);
                        }
                        else
                        {
                                enqueue(&q,x);
```

```
                }
                break;
        case 2:
                if(empty(&q))
                {
                        printf("queue is empty");
                        exit(0);
                }
                else
                {
                        x=dequeue(&q);
                        printf("element=%d",x);
                }
                break;
        case 3:
                print(&q);
                break;
        default:
                break;
        }
    }while(op!=4);
    getch();
}
void initialize(Q *p)
{
    p→R= - 1;
    p →F=  -1;
}
int empty (Q *P)
{
    if (p ->R == -1)
        return(1);
    else
        return(0);
}
```

```
int full (Q   *p)
{
    if(p→R == MAX-1);
            return (1);
    else
            return(0);
}
void enqueue (Q *p, int x)
{
    if (p→R == -1)                          /*empty queue  */
    {
            p→R =p→F=0;
            p→data  [p→R] =
x;
    }
    else
    {
            p→R=p →R+1;
            p →data [p→R]=x;
    }
}

int dequeue (Q *p)
{
    int x;
    x=p→data[p→F];
    if (p→R = = p→F)     /* last element */
    {
            p→R=-1;
            p→F=-1;
    }
    else
    {
            p→F = p→F+1;
    }
    return (x);
```

```
 }

void print (Q * p)
{
        int i;
        for (i=p→F; i <= p→R;i++)
        {
                printf("%d",p→data[i]);
        }
}
```

Circular Queues

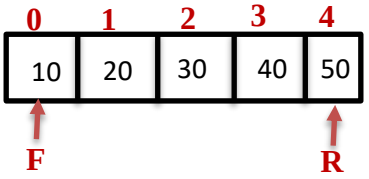
There is one potential problem with implementation of queue using a simple array.

The queue may appear to be full although there may be some space in the queue.

After insertion of five elements in the array

rear=4;

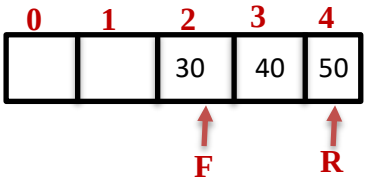
front=0 Queue is full



After two successive deletion

rear=4;

front=2 Queue is full



Queue appears to be full although there is some space in queue.

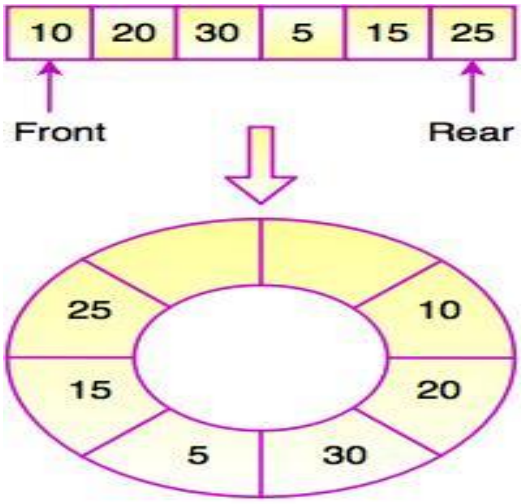


Fig. Circular Queue

Solution to this last position is connected back to first position.

Whenever rear gets to end of array it is wrapped around to the beginning.

Circular queue is linear data structure in which last element is connected to first to make circle.

Circular queue implementation

To give a circular movement inside array, whenever we go last element of the array it should come back to beginning of the array.

i	i=(i+1) % MAX		
0	(0+1) % 5	=1	
1	(1+1)%5	=2	
2	(2+1)%5	=3	
3	(3+1)%5	=4	
4	(4+1)%5	=0	i wrap around to beginning ‘0’

Operations on Circular Queue

```
typedef struct Q
{
        int data ;
        int R,F;
}Q;
```

'C' function to initialize()

```
void initialize(Q *p)
{
        p→R= -1;
        p→F= -1;
}
```

'C ' function to check whether queue is empty or not

```
int empty (Q *P)
{
    if (p→R == -1)
        return(1);
    else
        return(0);
}
```

'C' function to check whether queue is full or not .

```
int full (Q *p)
{
        if((p→R+1) % MAX ==p→F)
                return (1);
        else
                return(0);
}
```

'C' function for insertion in queue.

```
void enqueue (Q *p, int x)
{
        if (p→R==-1) /*empty queue  */
```

```
        {
                p→R =p→F=0;
        }
        else
        {
                p→R=(p→R+1)%MAX;
        }
        p→data[p→R] = x;
}
```

'C' function for deletion.

```
int dequeue (Q * p)
{
        int  x;
        x=p→data[p→F];
        if (p→R = = p→F)     /* last element */
                p→R = p→F=-1;
        else
                p→F = (p→F+1)%MAX ;
        return (x);
}
```

'C' function for printing a queue .

```
void print (Q * p)
{
        int i;
        i = p→F;      /* start from front */
        while (i! =p→R)
        {
                printf("\n %d", p→data[i]);
                i=(i+1)%MAX ;
        }
        printf("%d", p→data[p→R]);
    }
```

Write a program in c to implement a circular queue; the following operations should be performed by the program.

1. **Create 2. Insert 3. Delete 4. Display 5. Exit**

Write a program to implement circular queue using arrays. [10M MAY18, 10M MAY15, 10M DEC15]

```
#include<stdio.h>
#include<conio.h>
#define max 30
typedef struct Q
{
        int data[MAX] ;
        int F, R
}Q;
void initialize(Q *p)  ;
Int empty (Q *P) ;
Int full (Q   *p) ;
void enqueue (Q *p, int x);
int dequeue (Q *p);
void print (Q * p) ;
void main()
{
        int x, op, n;
        Q q;
        init(&q);
        do
        {
                Printf("1-create\n 2-insert\n 3-delete\n 4-print\n")
                printf("Enter your choice n");
                scanf("%d",&op);
                switch(op)
                {
                        case 1: printf("enter no of elements\n");
                                scanf("%d",&n);
                                printf("enter the data\n");
                                for(i=0;i<n;i++)
```

```
        {
                scanf("%d",&x);
                if (full(&q))
                {
                        printf("queue is full\n");
                        Exit(0);
                }
                else
                {
                        enqueue(&q,x);
                }
        }
        break;
Case 2:
        printf("enter the element to be inserted");
        scanf("%d",&x);
        if (full(&q))
        {
                printf("queue is full\n");
                Exit(0);
        }
        else
        {
                enqueue(&q,x);
        }
        break;
Case 3:
        if(empty(&q))
        {
                printf("queue is empty");
                Exit(0);
        }
        else
        {
                x=dequeue(&q);
```

```
                                printf("element=%d",x);
                        }
                        break;
                Case 4:
                        print(&q);
                        break;
                default:
                        break;
                }
        }while(op!=5);
        getch();
    }
    void initialize(Q *p)
    {
        p-→R= - 1;
        p →F=  -1;
    }
    int empty (Q *P)
    {
        if (p→R == -1)
                return(1);
        else
                return(0);
    }

    Int full (Q   *p)
    {
        if((p→R+1) % MAX ==p→F)
                return (1);
        else
                return(0);
    }
void enqueue (Q *p, int x)
{
        if (p→F== -1 )                  /*empty queue  */
```

```
        {
                p→R =p→F=0;
        }
        else
        {
                p→R=(p→R+1)%MAX;
        }
        p→data[p→R] =
x;
}

int dequeue (Q * p)
{
        int  x;
        x=p→data[p→F];
        if (p→R = = p→F)     /* last element */
                p→R = p→F=-1;
        else
                p→F = (p→F+1)%MAX ;
        return (X);
}

void print (Q * p)
{
        int i;
        i = p→ F;      /* start from front */
        while ( i! =p→R)
        {
                printf("%d", p→data[i]);
                i=(i+1)%MAX ;
        }
        printf("%d", p→data[p-R]);
    }
```

Priority Queue

- priority queue is an ordered list of elements.

- In priority queue services is provided on the basis of priority based on urgency of need.
- In a normal queue service is provided on the basis **"first come first served"** where as in priority queue service is provided on the basis of **priority.**
- An element with higher priority is processed before other element with lowest priority.
- An element with equal priority are process on "first-come-first-served" basis.
- example: -
  1) Hospital waiting room
  A patient with more fatal problem will be admitted before other patients.
  2) In long term scheduling of jobs in a computer.
  Short processes are given priority over long process to improve response time.

Priority queue can be implemented by

1. Circular array
2. Linked list

1. Implementation of priority queue using circular array

```
void initialize(Q *p)
{
    p→R= - 1;
    p →F=  -1;
}
int empty (Q *P)
{
    if (p→R == -1)
            return(1);
    else
            return(0);
}

Int full (Q   *p)
{
```

```
    if((p→R+1) % MAX ==p→F)
            return (1);
    else
            return(0);
}

void enqueue (queue *p, int x)
{
    int i;
    if (full (p))
    {
            printf("overflow\n");
            exit(0);
    }
    else
    {
            if (empty(p))
            {
                    P→rear = P→front = 0;
                    P→data[0]= x;
            }
            else
            {
                    i= p→rear;
                    while(x >p→data[i]);
                    {
                            p→data[(i+1)% max]= p→data[i];
                            i=(i-1+max)%max;
                            if ((i+1)%MAX = = P→ front)
                                    break;
                    }
                    i =(i+1)%max            /*insert*/
                    /*re-adjust error*/
                    p→data[i]= x;
                    p→rear=(p→rear + 1)%max;
            }
```

```
}

Int dequeue (Q * p)
{
        int  x;
        x=p→ data[p→F];
        if (p→R = = p→F)
        {
                p→F=-1;
                p→R=-1;
        }
        else
        {
                p→F = (p→F+1)%MAX ;
        }
        return (X);
}

void print (Q * p)
{
        int i;
        i = p→F;      /* start from front */
        while (i! =p→R)
        {
                printf("\n %d", p→data[i]);
                i=(i+1)%MAX ;
        }
        printf("%d", p→data[p→R]);
}
```

Write a c program to implement priority queue using array, the program should perform following operations. [12M DE 18]

i)Insert in a priority queue ii) Delete from a queue

iii) Display elements of a queue.

```
#include<stdio.h>
#include<conio.h>
#define max 30
typedef struct Q
{
        int data[MAX] ;
        int F, R
}Q;
void initialize(Q *p)  ;
Int empty (Q *P) ;
Int full (Q   *p) ;
void enqueue (Q *p, int x);
int dequeue (Q *p);
void print (Q * p) ;
void main()
{
        int x, op, n;
        Q q;
        init(&q);
        do
        {
                printf("1- Insert in a priority queue \n 2-delete\n 3-display\n 4-
            exit \n")
                printf("Enter your choice n");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:
                                printf("enter the element to be inserted");
                                scanf("%d",&x);
                                if (full(&q))
```

```
                {
                        printf("queue is full\n");
                        Exit(0);
                }
                else
                {
                        enqueue(&q,x);
                }
                break;
        case 2:
                if(empty(&q))
                {
                        printf("queue is empty");
                        Exit(0);
                }
                else
                {
                        x=dequeue(&q);
                        printf("element=%d",x);
                }
                break;
        case 3:
                print(&q);
                break;
        default:
                break;
    }
  }while(op!=4);
  getch();
}

void initialize(Q *p)
{
  p→R= - 1;
  p →F=  -1;
}
```

```
Int empty (Q *P)
{
    if (p→R == -1)
            return(1);
    else
            return(0);
}

Int full (Q   *p)
{
    if((p→R+1) % MAX ==p→F)
            return (1);
    else
            return(0);
}

void enqueue (queue *p, int x)
{
        int i;
        if (full (p))
        {
                printf("overflow\n");
                exit(0);
        }
        Else if (empty(p))
        {
                P→rear = P→front = 0;
                P→data[0]= x;
        }
        else
        {
                i= P→rear;
                while(x >p→data[i]);
                {
                        P→data[(i+1)% max]= P→data[i];
                        i=(i-1+max)%max;
                        if ((i+1)%MAX = = P→ front)
```

```
                        break;
                }
                i =(i+1)%max            /*insert*/
                /*re-adjust error*/
                P→data[i]= x;
                P→rear=(P→rear + 1)%max;
        }
}

Int dequeue (Q * p)
{
        int  x;
        x=p→ data[p→F];
        if (p→R = = p→F)
        {
                p→F=-1;
                p→R=-1;
        }
        else
        {
                p→F = (p→F+1)%MAX ;
        }
        return (X);
}

void print (Q * p)
{
        int i;
        i = p→F;      /* start from front */
        while (i! =p→R)
        {
                printf("\n %d", p→data[i]);
                i=(i+1)%MAX ;
        }
        printf("%d", p→data[p→R]);
}
```

Explain circular queue and double ended queue with example [10M DEC16].

Dequeue

It is double ended queue.

In a dequeue insertion as well as deletion can be carried out either at the rear end or front end.

Dequeue is classified into two types:

- Input restricted dequeue: -
    1) Insertion of an element at the rear.
    2) Deletion of an element from front.
    3) Deletion of an element from rear end.

- Output restricted dequeue: -
    1) Deletion of an element from front end.
    2) Insertion of an element at from rear end.
    3) Insertion of an element at the front end.

Operation on dequeue :-

- init() – make the queue empty.
- empty() – determine if queue is empty.
- full() – determine if queue is full.
- enqueueF() – insert an element at the front end.
- dequeueR() – delete rear element.
- dequeueF() – delete front element.
- printf() – print element of queue.

1. Write a program in c to implement Queue ADT using linked list to perform the following operations [10M MAY18]
   i) Insert a node in the queue ii)Delete a node from ququе
   iii) Display ququе elements

2. Write a program to implement circular queue using arrays. [10M MAY18, 10M MAY15, 10M DEC15]

3. Write a c program to implement priority queue using array, the program should perform following operations. [12M DE 18]
   i)Insert in a priority queue ii) Delete from a queue

   iii) Display elements of a queue.

4. Write a program to implement linear queue using array. [10 DEC17]

5. Give ADT for Queue data structure, discuss two applications of queue data structure.[5M MAY17]

6. Write a menu driven program in C to implement QUEUE ADT the program should perform following operations. [12M MAY16]
   v) Inserting element in the queue
   vi) Deleting an element from the queue.
   vii) Displaying the queue
   viii) Exiting the program

7. Explain circular queue and double ended queue with example [10M DEC16].

8. Write an algorithm to implement queue using array [10M MAY18,DEC18,DEC17 (IT)]

9. Write an algorithm to implement priority queue using array [10M May18 (IT)]