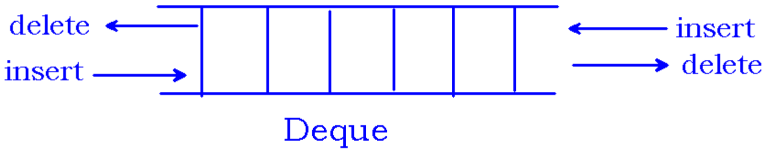
Double Ended Queue

Deque:

- It is a linear data structure.
- Using linear queue, we can insert element from one end and remove the elements from another end.
- In double ended queue, we can perform insert and delete operations from both the ends.
- It is also called Deque.



Operations of DEQUE:

1. create(): Define and initialize the queue with default values
2. isEmpty(): Returns a boolean true value if the queue is empty
3. isFull(): Determines whether the queue is full or not with boolean value
4. insertFront(): Inserting element at front of Deque
5. insertRear(): Insert an element at the rear end of the queue
6. deleteRear(): Delete the rear element
7. deleteFront(): Delete the front element
8. traverse(): Display elements of Queue

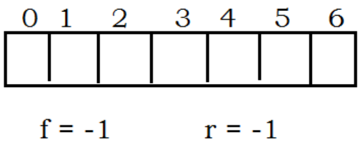
Declaration:

```
#define size 7
int deque[size];
int front=-1 , rear=-1 ;
```

Notes:

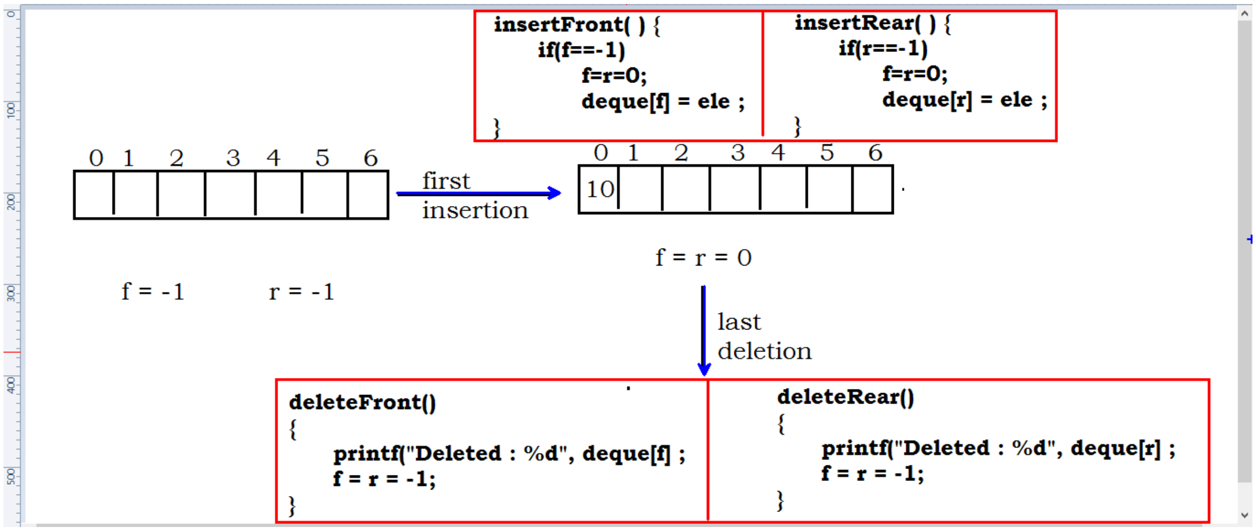
- Operating queue from front means – we use ‘front’ variable for insertions and deletions.
- Operating queue from rear means – we use ‘rear’ variable.

Empty Queue:



```
deleteFront( )
{
    if(f==-1)
        Empty...
}
```

```
deleteRear( )
{
    if(r==-1)
        Empty...
}
```



f = -1 r = -1

first insertion

```
insertFront( ) {
    if(f==-1)
        f=r=0;
        deque[f] = ele ;
}
```

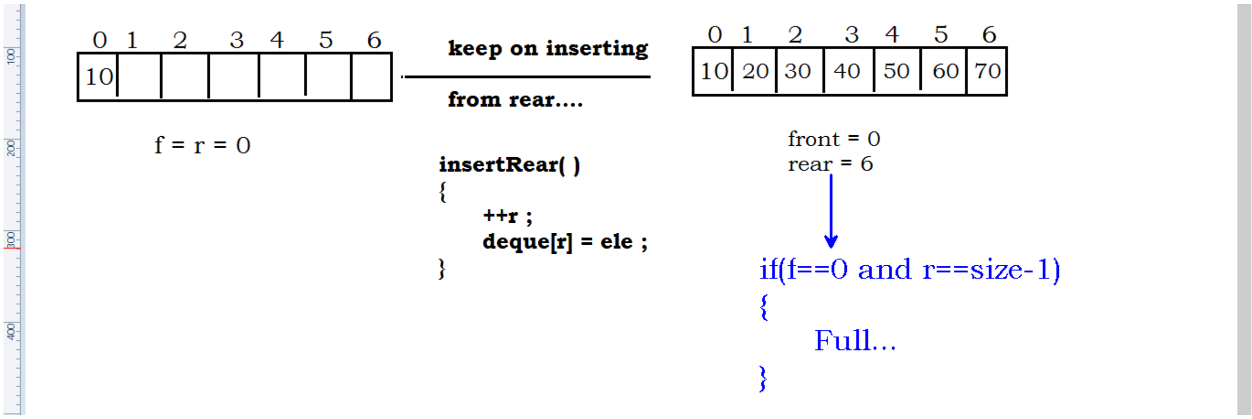
```
insertRear( ) {
    if(r==-1)
        f=r=0;
        deque[r] = ele ;
}
```

f = r = 0

last deletion

```
deleteFront()
{
    printf("Deleted : %d", deque[f] ;
    f = r = -1;
}
```

```
deleteRear()
{
    printf("Deleted : %d", deque[r] ;
    f = r = -1;
}
```



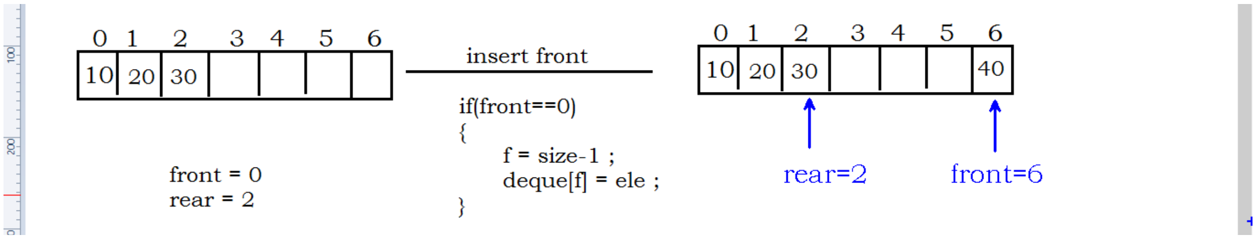
f = r = 0

keep on inserting from rear....

```
insertRear( )
{
    ++r ;
    deque[r] = ele ;
}
```

front = 0
rear = 6

```
if(f==0 and r==size-1)
{
    Full...
}
```

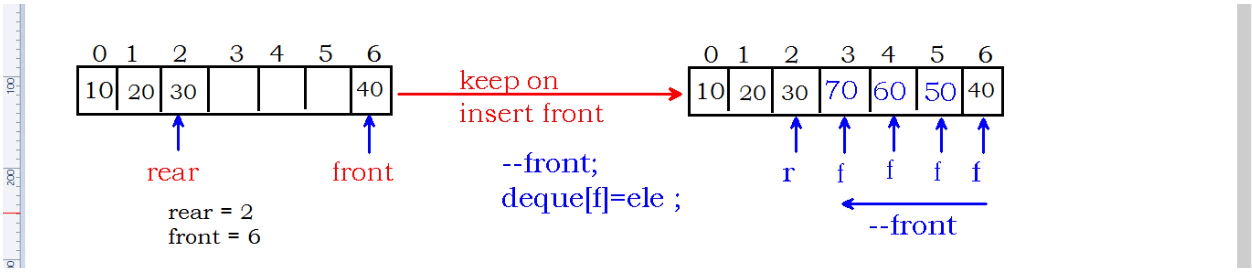


front = 0
rear = 2

insert front

```
if(front==0)
{
    f = size-1 ;
    deque[f] = ele ;
}
```

rear=2 front=6

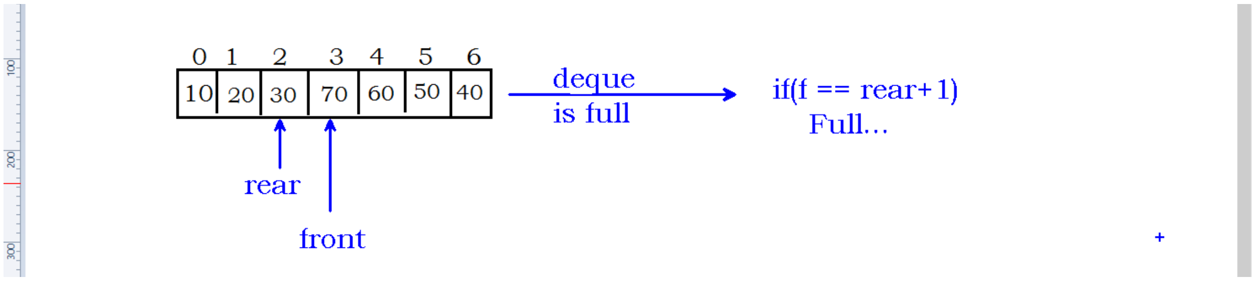


0	1	2	3	4	5	6
10	20	30				40

rear = 2
front = 6

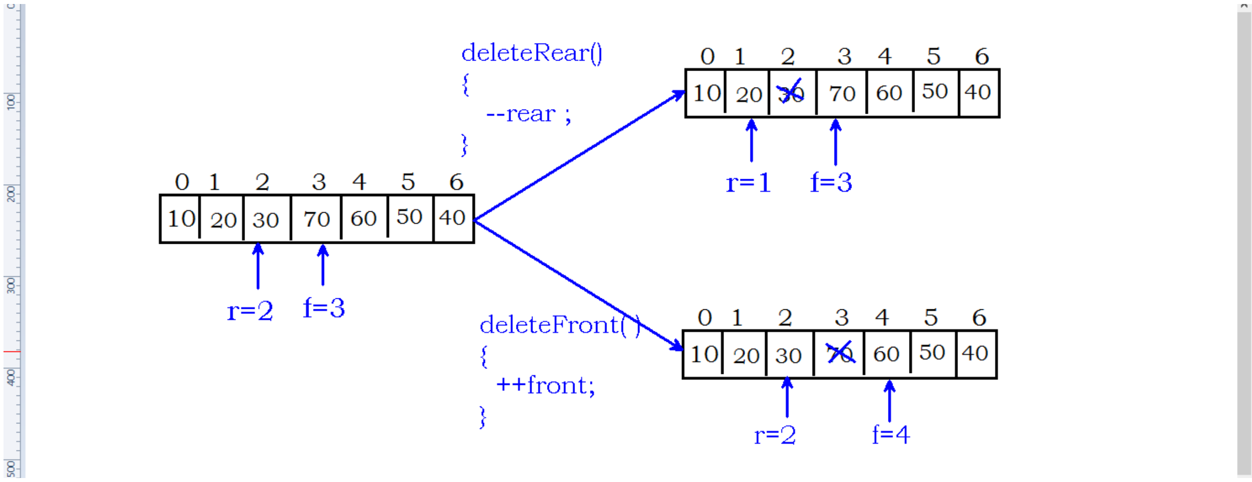
```
--front;
deque[f]=ele ;
```

0	1	2	3	4	5	6
10	20	30	70	60	50	40



0	1	2	3	4	5	6
10	20	30	70	60	50	40

```
if(f == rear+1)
   Full...
```



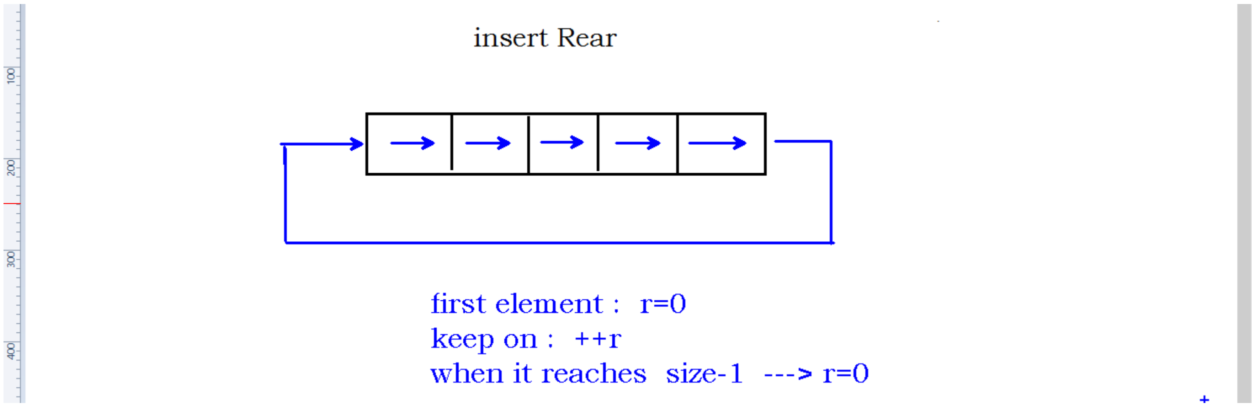
0	1	2	3	4	5	6
10	20	30	70	60	50	40

```
deleteRear()
{
  --rear ;
}
```

0	1	2	3	4	5	6
10	20	~~30~~	70	60	50	40

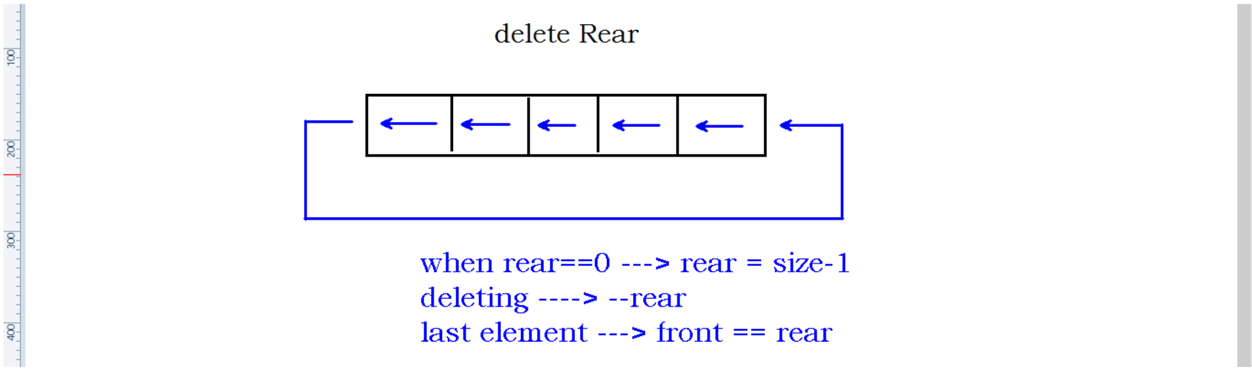
```
deleteFront( )
{
  ++front;
}
```

0	1	2	3	4	5	6
10	20	30	~~70~~	60	50	40



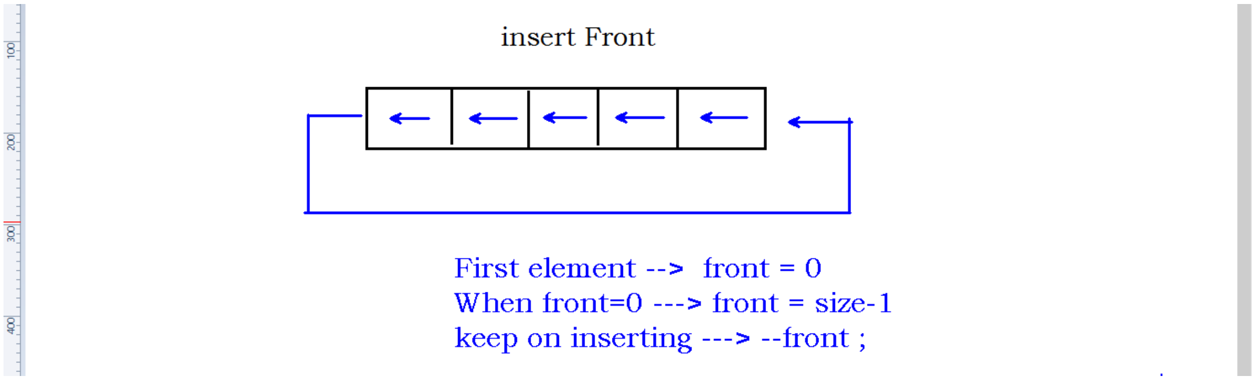
insert Rear

first element : r=0
keep on : ++r
when it reaches size-1 ---> r=0



delete Rear

when rear==0 ---> rear = size-1
deleting ----> --rear
last element ---> front == rear



insert Front

First element --> front = 0
When front=0 ---> front = size-1
keep on inserting ---> --front ;

```
#include<stdio.h>
#define SIZE 5
int deque[SIZE] , front=-1 , rear=-1 ;
void insertFront(int);
void insertRear(int);
void deleteFront(void);
void deleteRear(void);
void display(void);
int isOverflow(void);
int isUnderflow(void);
void main()
{
        int choice , ele ;
```

```
    while(1)
    {
        printf("/*Deque operations**/\n");
        printf("1.InsertFront\n");
        printf("2.InsertRear\n");
        printf("3.DeleteFront\n");
        printf("4.DeleteRear\n");
        printf("5.Display\n");
        printf("6.Quit\n");

        printf("Enter your choice : ");
        scanf("%d", &choice);

        if(choice==1 || choice==2)
        {
            printf("Enter element to insert : ");
            scanf("%d", &ele);
        }

        switch(choice)
        {
            case 1          :   insertFront(ele);
                                break ;

            case 2          :   insertRear(ele);
                                break ;

            case 3          :   deleteFront();
                                break ;

            case 4          :   deleteRear();
                                break ;

            case 5          :   display();
                                break ;

            case 6          :   exit(1);
            defualt :       printf("Invalid choice...\n\n");
        }
    }
}
void insertFront(int ele)
{
    if(isOverflow())
    {
        printf("Deque is Full \n\n");
    }
    else
    {
        if(front==-1)
        {
```

```
                        front = rear = 0;
                }
                else if(front==0)
                {
                        front = SIZE-1 ;
                }
                else
                {
                        front--;
                }
                deque[front] = ele ;
        }
}
void insertRear(int ele)
{
        if(isOverflow())
        {
                printf("Deque is Full \n\n");
        }
        else
        {
                if(rear==-1)
                {
                        front = rear = 0 ;
                }
                else if(rear==SIZE-1)
                {
                        rear = 0 ;
                }
                else
                {
                        rear++ ;
                }
                deque[rear] = ele ;
        }
}
void deleteFront(void)
{
        if(isUnderflow())
        {
                printf("Deque is Empty \n\n");
        }
        else
        {
                printf("Deleted : %d \n\n", deque[front]);
                if(front==rear)
                {
                        front = rear = -1 ;
                }
                else if(front == SIZE-1)
                {
```

```
            front = 0 ;
        }
        else
        {
            front++ ;
        }
    }
}
void deleteRear(void)
{
    if(isUnderflow())
    {
        printf("Deque is Empty \n\n");
    }
    else
    {
        printf("Deleted : %d \n\n", deque[rear]);
        if(front==rear)
        {
            front = rear = -1 ;
        }
        else if(rear == 0)
        {
            rear = SIZE-1 ;
        }
        else
        {
            rear-- ;
        }
    }
}
void display()
{
    int i;
    if(front == -1)
    {
        printf("No elements to display\n\n");
    }
    else if(front<=rear)
    {
        for(i=front ; i<=rear ; i++)
        {
            printf("Element %d : %d \n", i+1, deque[i]);
        }
        printf("\n");
    }
    else
    {
        for(i=front ; i<=SIZE-1; i++)
        {
            printf("Element %d : %d \n",i+1,deque[i]);
```

```
                }
                for(i=0 ; i<=rear ; i++)
                {
                        printf("Element %d : %d \n",i+1,deque[i]);
                }
                printf("\n");
        }
}
int isOverflow(void)
{
        if((front==rear+1) || (front==0 && rear==SIZE-1))
                return 1 ;
        else
                return 0 ;
}
int isUnderflow(void)
{
        if(front==-1 || rear==-1)
                return 1 ;
        else
                return 0 ;
}
```

Implementing DEQUE using pointers:

- We represent the data using structure in this implementation.
- We access elements using point to structure format.
- Structure is an object; hence it is protected compare to regular implementation.

```
#include<stdio.h>
#include<process.h>
#define SIZE 6

typedef struct DoubleEndedQueue
{
        int data[SIZE];
        int rear,front;
}deque;

deque q ;

int isEmpty(deque *p);
int isFull(deque *p);
void insertRear(deque *p,int x);
void insertFront(deque *p,int x);
int deleteFront(deque *p);
int deleteRear(deque *p);
void print(deque *p);

void main()
```

```
{
    int i,x,ch,n;
    q.front=-1 ;
    q.rear=-1 ;
    while(1)
    {
        printf("\n1.Insert rear :");
        printf("\n2.Insert front :");
        printf("\n3.Delete rear :");
        printf("\n4.Delete front :");
        printf("\n5.Display :");
        printf("\n6.Exit\n");
        printf("\nEnter your choice :");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1: printf("Enter element to insert : ");
                    scanf("%d",&x);
                    if(isFull(&q))
                        printf("Queue is isFull \n\n");
                    else
                        insertRear(&q,x);
                    break;

            case 2: printf("Enter element to insert :");
                    scanf("%d",&x);
                    if(isFull(&q))
                        printf("Queue is isFull\n\n");
                    else
                        insertFront(&q,x);
                    break;

            case 3: if(isEmpty(&q))
                    printf("Queue is isEmpty\n\n");
                    else
                    {
                        x=deleteRear(&q);
                        printf("Deleted item is : %d\n\n",x);
                    }
                    break;

            case 4: if(isEmpty(&q))
                    printf("Queue is isEmpty\n\n");
                    else
                    {
                        x=deleteFront(&q);
                        printf("Deleted item is : %d\n\n",x);
                    }
                    break;
            case 5:print(&q);
                    break;

            case 6: exit(1);

            default: printf("Invalid choice \n\n");
        }
    }
}

int isEmpty(deque *P)
{
    if(P->rear==-1)
        return(1);

    return(0);
}

int isFull(deque *P)
{
    if((P->rear+1)%SIZE==P->front)
        return(1);

    return(0);
}

void insertRear(deque *P,int x)
{
    if(isEmpty(P))
    {
        P->rear=0;
        P->front=0;
        P->data[0]=x;
    }
    else
    {
        P->rear=(P->rear+1)%SIZE;
        P->data[P->rear]=x;
    }
}

void insertFront(deque *P,int x)
{
    if(isEmpty(P))
    {
        P->rear=0;
        P->front=0;
        P->data[0]=x;
    }
    else
    {
```

```
                P->front=(P->front-1+SIZE)%SIZE;
                P->data[P->front]=x;
        }
}

int deleteFront(deque *P)
{
        int x;
        x=P->data[P->front];
        if(P->rear==P->front)
        {
                P->front=-1;
                P->rear=-1;
        }
        else
        {
                P->front=(P->front+1)%SIZE;
        }

        return(x);
}

int deleteRear(deque *P)
{
        int x;
        x=P->data[P->rear];
        if(P->rear==P->front)
        {
                P->front=-1;
                P->rear=-1;
        }
        else
        {
                P->rear=(P->rear-1+SIZE)%SIZE;
        }
        return(x);
}

void print(deque *P)
{
        int i;
        if(isEmpty(P))
        {
                printf("\nQueue is isEmpty!!");
                exit(0);
        }
        i=P->front;
        while(i!=P->rear)
        {
                printf("\n%d",P->data[i]);
                i=(i+1)%SIZE;
        }
        printf("\n%d\n",P->data[P->rear]);
}
```