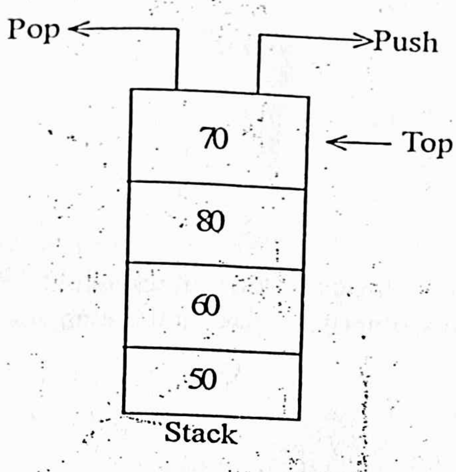
Raju

UNIT-II

STACK

A "STACK" is a linear collection of elements, where we can perform both the insertion and deletion. The process of insertion is called "PUSH" and the process of deletion is called "POP". A "stack" must viewed as a vertical array. Here both the insertion and deletion will take from one end called "TOP". It follows the "LIFO"(Last in first out) that means the last inserted element will be deleted first.



ALGORITHM:-

PUSH:-The process of inserting an element into the stack is called as "push", before inserting an element into a stack, we have to check whether the stack is full or not. if"TOP=MAX" then we say stack is full otherwise increment the top by one and push the element into STACK[Top]

```
Push(s, x, top, max):
{
"s" is a stack
"x" is an element to be insert
"TOP" is an index of last element
"MAX" is the maximum size of STACK
Step-1: start
Step-2: if (Top== MAX) then
          Write ("stack is full");
        Else
         (a)Top=Top+1
         (b)S[Top]=x
Step-3: stop
}
```

Pop:- The process of deleting an element from the top of the stack is called POP. Before deleting the element from the stack, we must test whether stack is empty or not . If TOP =0 then stack is said to be empty and then the top is decrement by 1.

```
POP(S, X, TOP, MAX)
{
"S" is a stack
"X" is where deleted element TO BE PLACED
"TOP" is an index of last element
"MAX" is the size
Step-1: START
Step-2:  if (TOP == 0 )then
                    Write ("stack is empty")
         else
                    (C) x=s[Top];
                    (D) Top--;
Step -3: stop
}
```

Display:-For displaying elements in the stack, first check whether stack is empty or not.if it is empty then print stack is empty otherwise display the elements.
Algorithem:-Display()

```
{
Step 1: START
Step 2:if top==-1 then
         write " stack is empty"
Step 3: else
          for(i=0,i<=top;i++)
                write (s[i]);
Step 5: STOP
}
```

Programe:- (To Implementation Stack using Arrays) (or Write a Java Program for Stack using Arrays)

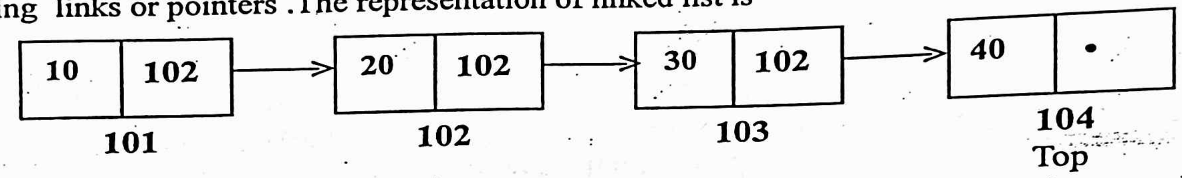
```
import java.io.*;
import java.util.*;
class mystack
{
int top;
int n=5;
int []s=new int[5];
mystack()
{
top=-1;
}
void push(int x)
{
if(top==n-1)
System.out.println("Stack is Full");
else
{
top++;
s[top]=x;
```

```
}
}
void pop()
{
if(top==-1)
{
System.out.println("Stack is Empty");
}
else
{
int x;
x=s[top];
top=top-1;
}
}
void display()
{
int i;
if(top==-1)
{
System.out.println("Stack is Empty");
}
else
{
for(i=0;i<=top;i++)
System.out.println("nos="+s[i]);
}
}
}
class stack
{
public static void main(String arg[])throws IOException
{
int x,ch;
mystack ob=new mystack();
do
{
System.out.println("1.PUSH");
System.out.println("2.POP");
System.out.println("3.DISPLAY");
System.out.println("4.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
```

```
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.push(x);
break;
case 2:
ob.pop();
break;
case 3:
ob.display();
break;
case 4:
System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=4);
}
}
```

STACK WITH LINKED LIST

Linked stack is dynamic data structure because to develop the linked stack by using links or pointers .The representation of linked list is



In the above stack contains four nodes like 10, 20, 30and 40. Here the 40 is top node because it is most recently element. Manipulate the linked stack by using 3 types of operations like.

1. Push operation
2. Pop operation
3. Traverse operation

Push operation :-Suppose if you want to insert an element into the stack called "push operation". In this we first check the condition stack is empty (or) not. If it is empty then insert a new node and assign null pointer to the link otherwise insert a new node and assign next node address to the link.

ALGORITHM:-Push (num)

```
Step 1:-  START
Step 2:-  if(top=null)then
            t=new node
            t.data-num
            t.link=null
            top=t
      else
            t=new node
            t.data =num
             t.link-top=t
           top=t
Step 3:-  STOP
```

Pop :-Suppose if you want to delete an element from the stack called "pop operation". In this, we first check the condition stack is empty (or) not. If it is empty then display a message "stack is empty". Otherwise insert a new node and assign next node address to the top.

ALGORITHM: -

```
Step 1:-START
Step 2:- if (top=null)
              Then stack is empty
         else
           t=top
            x=top.data;
           top=top.link
          Delete(t);
Step 3:-  STOP
```

Traverse :-Suppose if you want visit each and every element from the stack called "Traverse". In this to display the list of elements from first element to last element.

ALGORITHM: -

```
Step 1:-START
Step 2:- Set a=top
Step 3:- Repeat the  step 4 until a not equal to NULL
Step 4:  While(a!=NULL)
                print a.data
               a=a.link
Step 4 :-STOP
```

```
import java.io.*;
import java.util.*;
class node
{
int ele;
node link;
}
class mystack
{
node top;
mystack()
{
top=null;
}
void push(int x)
{
node t=new node();
t.ele=x;
if(top==null)
t.link=null;
else
t.link=top;
top=t;
}

void pop()
{
int x;
if(top==null)
{
System.out.println("Stack is Empty");
}
else
{
x=top.ele;
top=top.link;
}
}
void display()
{
node p;
if(top==null)
{
System.out.println("Stack is Empty");
}
else
{
for(p=top;p!=null;p=p.link)
System.out.println("nos="+p.ele);
}
}
}
class linkstack
{
public static void main(String arg[])throws IOException
{
int x,ch;
mystack ob=new mystack();
do
{
System.out.println("1.PUSH");
System.out.println("2.POP");
System.out.println("3.DISPLAY");
System.out.println("4.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.push(x);
break;
case 2:ob.pop();
break;
case 3:ob.display();
break;
case 4:System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=4);
}
}
```

Applications of stacks:-

Three applications of stacks are presented here. These examples are central to many activities that a computer must do and deserve time spent with them.

1.Expression evaluation

2. Balancing of symbols

3. Backtracking(game playing, finding paths)

4. Infix to postfix conversion

5. Evaluation of postfix expression

6. Memory management, run-time enveronment for nested languages features.

7. Implementing function calls

8. Finding of spans

9. Matching tags in HTML and XML.

Evaluation of arithmetic expression:-Expression is a combination of operators and operands.An expression contains only arithematic expression mainly 3 types

* Inpex:- In Infixoperator is placed between the operands.

Ex:- a+b, a+(b*c)/d etc

*Prefix:-In Prefix operator is placed before the operands.

Ex:- +ab,+/d*cba etc

*Postfix :- In Postfix operator is placed after the operands.

Ex:-ab+,abc*d/+ etc

Generally computer can accepted infix expression and convert that expression in prefix (Or) postfix expression.The following algorithem is used to convert the infix expression into prefix (Or) postfix.

Algorithm:- Infix=>prefix (Or) postfix

Step 1: Start
Step 2: Scan the infix expression one-buy-one until you reach end of the expression
Step 3: When an operand is scanedthen to place in postfix (Or) prefix
Step 4: when an operator (Or) left brace is scaned then to place in stack.
Step 5: when right brace is scanned, pop all the operators from the stack until left brace is scanned
Step 6: you reach end of the expression.
Step 7 : Stop.

To convert Infix expression into postfix:-

Ex:- Infix= (a + (b * c)/d)

Scan	**Stack**	**Postfix**
(	(	
a	(	a
+	(+	a

(	(+ (	a
b	(+ (	ab
*	(+ (*	ab
c	(+ (*	abc
)	(+ (*)	abc *
/	(+ /	abc *
d	(+ /	abc * d
)	(+ /)	abc * d / +

Evaluate:-

Infix	Postfix
a + (b * c)/d	abc * d /+
= 2 + (2* 2)/2	= 222 * 2 /+
= 2 +4/2	= 242/+
= 2+2	= 22+
= 4	= 4

To convert Infix expression into prefix:-

Ex:- Infix= (a + (b * c)/d)

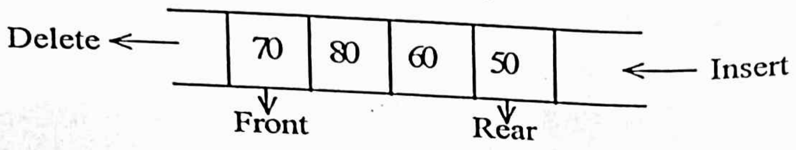
Scan	Stack	Prefix
(	(	
a	(	a
+	(+	a
(	(+ (	a
b	(+ (	ba
*	(+ (*	ba
c	(+ (*	cba
)	(+ (*)	*cba
/	(+ /	*cba
d	(+ /	d * cba
)	(+ /)	+ / d * cba

Evaluate:-

Infix	Prefix
a + (b * c)/d	+/ d*cba
= 2 + (2* 2)/2	= +/2*222
= 2 +4/2	= /+242
= 2+2	= +22
= 4	= 4

QUEUE

A" QUEUE" is a linear list of elements where we can perform the insertion at one end and deletion at another end.The insertion at RARE end and deletion at FRONT end. rare pointer is used insert the element and front pointer is uded to delete the element. It follows FIFO(first in first out), that means the first inserted element will be deleted first



INSERT: Before inserting an element in to the queue we have to check the condition whether it is full or not. for this , the condition is if MAX=REAR then we will say the QUEUE is full Otherwise increment RARE by one & push that element into Q[rear].

Algorithm:-

```
Insert (Q , X, REAR, FRONT, MAX)
{
"Q" is a queue
"x" is an element to be insert
"REAR" is an index of last element
"FRONT" is an index of first element
"MAX" is the maximum size
Step 1: start
Step 2: if MAX=REAR Then
            Write (`queue is full`)
        Else
        (a)REAR =REAR+1
        (b)Q [REAR]=X,
STEP 3: STOP
}
```

Deletion: - Before deleting an element in to the queue, we have to check the condition whether it is empty or not .for this, the condition is if front=0 then we will say the QUEUE is empty. Otherwise remove the front element and increment FRONT by one.

Algorithm:-

```
Delete ( Q ,X , REAR, FRONT, MAX)
{
`Q` is a queue
`X` is a deleted element will store
`REAR` is an index first element
`MAX` is the maximum size

}
```

```
Step 1:     START

Step 2:     if front = 0 then
            Write (`queue is empty`)
            else
            (a) x=Q [FRONT]
            (b) front=front+1;
Step 3:     STOP
```

List : it is used to display the element list in queue.

<u>Algorithm :</u>

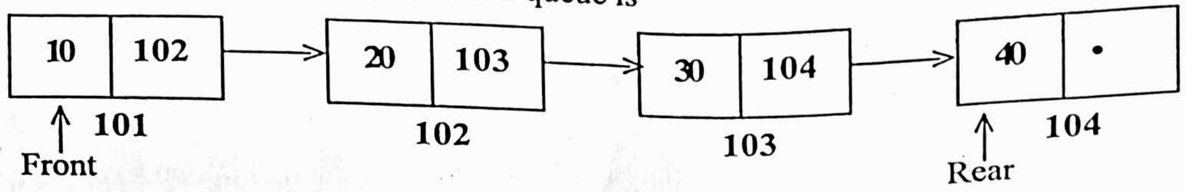
```
List (Q, MAX, REAR, FRONT)
{
`Q` is a Queue
`MAX` is the maximum size
`REAR` is an index of last element
`FRONT` is an index first element
}
Step 1:     START
Step 2:      for (i=front; i<=rear; i++)
            Write (Q[i]);
Step 3 :     STOP
```

```
import java.io.*;
import java.util.*;
class myqueue
{
int front;
int rear;
int n=5;
int []q=new int[5];
myqueue()
{
front=rear=-1;
}
void insert(int x)
{
if(rear==n-1)
{
System.out.println("Queue is Full");
}
else if(rear==-1)
{
front=rear=0;
}
else
rear=rear+1;
q[rear]=x;
}
void delete()
{
int x;
if(front==-1||rear==-1)
System.out.println("Queue is Empty");
else if(front==rear)
{
x=q[front];
front=rear=-1;
}
else
{
x=q[front];
front=front+1;
}
}

void display()
{
int i;
if(front==-1)
{
System.out.println("Queue is Empty");
}
else
{
for(i=front;i<=rear;i++)
System.out.println("nos="+q[i]);
}
}
}
class queue
{
public static void main(String arg[])throws IOException
{
int x,ch;
myqueue ob=new myqueue();
do
{
System.out.println("1.INSERT");
System.out.println("2.DELETE");
System.out.println("3.DISPLAY");
System.out.println("4.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.insert(x);
break;
case 2:
ob.delete();
break;
case 3:
ob.display();
break;
case 4:
System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=4);
}
}
```

QUEUE WITH LINKED LIST:-

Linked queue is a dynamic data structure because to develop the queue by using linked list. The representation of linked queue is



In the above queue contains 4 elements like 10, 20, 30 and 40. On that 10 is FRONT node and 40 is REAR node. To develop the linked queue by using 3 types of operations like

a) Insertion
b) Deletion
c) Traverse

Insertion:- suppose if you want to insert an element into the queue called insertion. In this we first check the condition queue is empty or not. If it is empty then create a new node and assign null pointer to the link and also assign that node address to the front and rear. otherwise to create a new node and assign that address to the previous node link and rare.

ALGORITHM:- Insertion(num)

```
Step 1: START
step 2 :if (REAR=NULL) then
            t=new node
            t.date=num
            t.link=null
            front=rear=t
            else
            t=new node
            t.data=num
             t.link=null
            rear.link=t
            rear=t
step 3:-STOP
```

Deletion:- supose if you want to delete an element into the queue called Deletion. In this we first check the condition queue is empty or not. If it is empty then to display the message "queue is empty ". Otherwise to delete front element and next move the front to the next node.

Algorithm:- Deletion()

```
Step 1: START
step 2 : if (FRONT=NULL) then
                write (queue is empty)
         else
                t=front
                x=front.data;
                front=front.link
                Delete(t)
Step 3:- STOP
```

Traverse:-visit and display the each and every element from the queue called as Traverse. In this we traverse the element from first element to last element.

Algorithm:-Traverse()

```
Step1:-START
Step 2:-set a=front
Step 3:-Repeate the steps 4 & 5
Step 4:-While (a!=null)
        print a.data
Step 5:-a=a.link
Step 4:-STOP
```

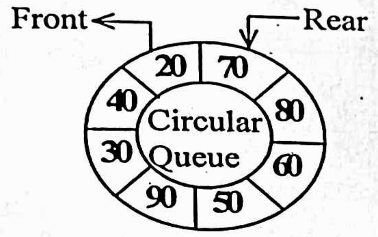
```
import java.io.*;
import java.util.*;
class node
{
int ele;
node link;
}
class myqueue
{
node front,rear;
myqueue()
{
front=rear=null;
}
void insert(int x)
{
node t=new node();
t.ele=x;
t.link=null;
if(rear==null)
front=rear=t;
else
{
rear.link=t;
rear=t;
}
}
void delete()
{
int x;
if(front==null)
{
System.out.println("Queue is Empty");
}
x=front.ele;
if(front.link==null)
front=rear=null;
else
front=front.link;
}
void display()
{
node p;
if(front==null)
{
System.out.println("Queue is Empty");
}
else
{
for(p=front;p!=null;p=p.link)
System.out.println("nos="+p.ele);
}
}
}
class linkqueue
{
public static void main(String arg[])throws IOException
{
int x,ch;
myqueue ob=new myqueue();
do
{
System.out.println("1.INSERT");
System.out.println("2.DELETE");
System.out.println("3.DISPLAY");
System.out.println("4.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.insert(x);
break;
case 2:
ob.delete();
break;
case 3:
ob.display();
break;
case 4:
System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=4);
}
}
```

CIRCULAR QUEUE

A linear queue is inefficient in certain cases. Once the queue is full we cannot perform any insertion even after deleting some elements from FRONT.This in efficincy can be over come by viewing the queue as circular.

The circular queue is also a queue in which it has two ends called "FRONT" and "REAR".The elements are inserted from REAR and deleted from FRONT.so it also follow the concept of FIFO. Let `Q` is queue with `n` elements.

Then Q[1] is front element and Q[n] is rear element in a circular queue.all positions are treated as circular behavior.



OPERATIONS:-We can perform the insertion and deletion operations on a circular queue.

INSERTION:- Adding a new element into the circular queue at rear position is called insertion. Before inserting an element, we must check the queue is full or not . if it is full then print "queue is full" otherwise increment REAR by one and put the element in Q[REAR].

```
ALOGORITHM:-Insert(Q,max, FRONT, REAR, X)
        {
            `Q`is circular
              `front` is an index of first element;
              `rear` is an index of last element;
                `x` is an element to be inserted;
              ``MAX`` is the size
        }
Step 1:-START
Step 2:-READ X;
Step 3:-if (REAR==MAX-1&FRONT==0)
            write ("queue is full")
            else if (REAR==MAX-1&&FRONT>0)
            FRONT-REAR=0
Step 4 : -  STOP;
```

DELETION:- Before deleting an element the QUEUE must be tested who it is empty or not . If (FRONT=0)the QUEUE is said be empty. If QUEUE is not empty ,then the element in the first positon is removed and increment FRONT by 1

ALGORITHM:-

Delete (Q, MAX, front, x, rear)

{

```
`Q`is queue;
`MAX ` is the size;
`Front` is an index of first element;
`Rear` is an index of last element;
`X` is where deleted element will be stored;
}
Step 1 :-       start
Step 2:-        if (front=-1|| rear=-1)
                Write ("queue is empty");
                else if (front==rear)
                X=Q[front];
                Front=rear=-1
                else if front == MAX-1;
                X=q[front]
                Front = 0
                else
                x=q[front]
                front++;
step 3:-        stop
```

LIST: Displaying all the elements in a queue is called list.

ALGORITHM:-

```
List (Q, MAX , front, rear)
{
`Q` is queue
`MAX` is the size;
`Front` is an index of first element;
`Rear` is an index of last element;
}
Step 1: -       START
Step 2:-        for ( i =front; i <= rear ; i++)
                Write (a[ i ]);
Step 3:-        stop
```

Program:-

```
import java.io.*;
import java.util.*;
class myqueue
{
int front;
int rear;
int n=5;
int []q=new int[5];
myqueue()
{
```

```
front=rear=-1;
}
void insert(int x)
{
if(rear==n-1&&front==0)
{
System.out.println("Queue is Full");
}
else if(rear==n-1&&front>0)
{
rear=0;
}
else if(front==-1||rear==-1)
{
front=rear=0;
}
rear=rear+1;
q[rear]=x;
}
void delete()
{
int x;
if(front==-1||rear==-1)
System.out.println("Queue is Empty");
else if(front==rear)
{
x=q[front];
front=rear=-1;
}
else if(front==n-1)
{
x=q[front];
front=0;
}
else
{
x=q[front];
front=front+1;
}
}
void display()
{
int i;
if(front==-1||rear==-1)
{
System.out.println("Queue is Empty");
else
{
for(i=front;i<=rear;i++)
System.out.println("nos="+q[i]);
}
}
}
class cirqueue
{
public static void main(String arg[])throws IOException
{
int x,ch;
myqueue ob=new myqueue();
do
{
System.out.println("1.INSERT");
System.out.println("2.DELETE");
System.out.println("3.DISPLAY");
System.out.println("4.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.insert(x);
break;
case 2:
ob.delete();
break;
case 3:
ob.display();
break;
case 4:
System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=4);
}
}
```

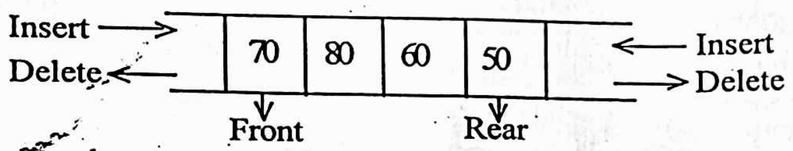
Application of Queue:-

1.printing job management i.e jobs to a network printer is enqueued so that the earlier job will be print first.

2.Queues used in Breadth first search.
3.Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
4.packet forwarding in router.
5.round robin scheduling.

DeQueue (or) Deque (Double ended Queue) :-

DeQueue is a data structure in which elements may be added to or deleted from the front or the rear. Like an ordinary queue, a double-ended queue is a data structure it supports the following operations: enq_front, enq_back, deq_front, deq_back, and empty. Dequeue can be behave like a queue by using only enq_front and deq_front , and behaves like a stack by using only enq_front and deq_rear.
The DeQueue is represented as follows.



DeQueue can be represented in two ways they are
1) Input restricted DeQueue
2) output restricted DeQueue
The out put restricted Dequeue allows deletions from only one end and input restricted Dequeue allow insertions at only one end.
The DeQueue can be constructed in two ways they are
2) Using array 2. using linked list

Algorithm to add an element into DeQueue :-

Assumptions: pointer f,r and initial values are -1,-1
Q[] is an array
max represent the size of a queue

enq_front:-
```
step1. Start
step2. Check the queue is full or not as if (f<>
step3. If false update the pointer f as f= f-1
step4. Insert the element at pointer f as Q[f] = element
step5. Stop
```
enq_back:-
```
step1. Start
step2. Check the queue is full or not as if (r == max-1) if yes queue is full
step3. If false update the pointer r as r= r+1
step4. Insert the element at pointer  as Q[r] = element
step5. Stop
```
Algorithm to delete an element from the DeQueue

deq_front:-
```
step1. Start
```

```
step2. Check the queue is empty or not as if (f == r) if yes queue is empty
step3. If false update pointer f as f = f+1 and delete element at position f as element =
Q[f]
step4. If ( f== r) reset pointer f and r as f=r=-1
step5. Stop
```

deq back:-

```
step1. Start
step2. Check the queue is empty or not as if (f == r) if yes queue is empty
step3. If false delete element at position r as element = Q[r]
step4. Update pointer r as r = r-1
step5. If (f == r ) reset pointer f and r as f = r= -1
step6. Stop
```

Priority Queue

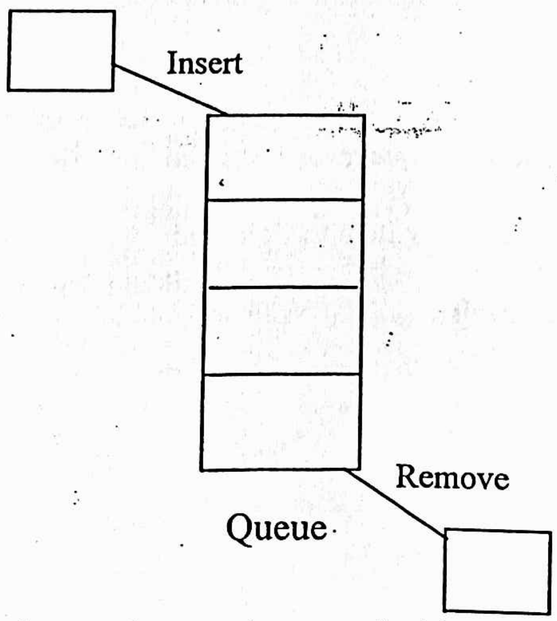
Priority Queue is more specialized data structure than Queue. Like ordinary queue, priority queue has same method but with a major difference. In Priority queue items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear or vice versa. So we're assigned priority to item based on its key value. Lower the value, higher the priority. Following are the principal methods of a Priority Queue.

Basic Operations:-

insert / enqueue - add an item to the rear of the queue.

remove / dequeue - remove an item from the front of the queue.

Priority Queue Representation



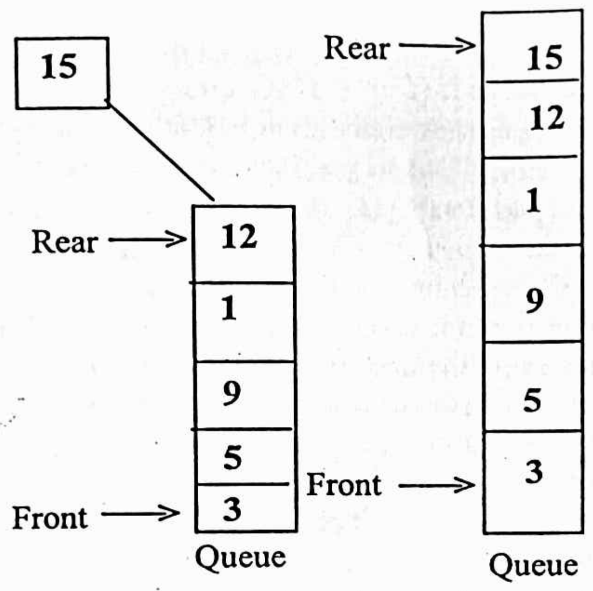
We're going to implement Queue using array in this article. There is few more operations supported by queue which are following.

Peek - get the element at front of the queue.

isFull - check if queue is full.
isEmpty - check if queue is empty.

Insert / Enqueue Operation:-

Whenever an element is inserted into queue, priority queue inserts the item according to its order. Here we're assuming that data with high value has low priority.



One item inserted rear end

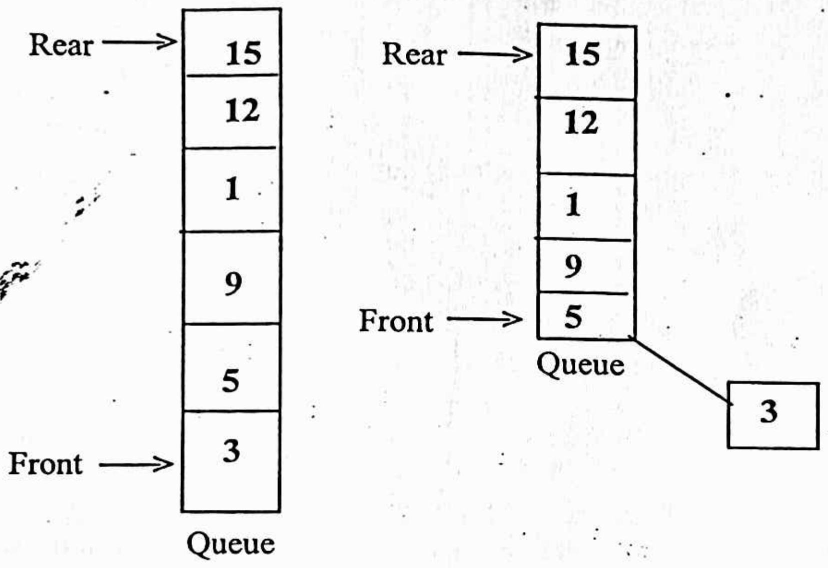
Insert Operation:-

```
void insert(int data)
{
  int i = 0;
  if(!isFull()){
    //if queue is empty, insert the data
    if(itemCount == 0){
      intArray[itemCount++] = data;
    }else{
      //start from the right end of the queue
      for(i = itemCount - 1; i >= 0; i-- ){
        // if data is larger, shift existing item to right end
        if(data > intArray[i]){
          intArray[i+1] = intArray[i];
        }else{
          break;
        }
      }
      // insert the data
      intArray[i+1] = data;
      itemCount++;
    }
  }
}
```

Remove / Dequeue Operation

Whenever an element is to be removed from queue, queue get the element using item count. Once element is removed. Item count is reduced by one.

Queue Remove Operation
```
int removeData()
{
  return intArray[--itemCount];
}
```



One item removed from front

Priority queue applications:-

Priority queues have many applications and below are few of them

1,Data compression

2.shortest path algorithem

3.Minimum spanning tree algorithem

4.Event driven simulation