

UNIT-2

Stacks & Queues

1Q. Define Stack. Explain stack operations (or)

Define Stack. Explain implementation of stack using arrays.

----- 10 Marks (Very Imp)

Ans: Stack ADT : A Stack is an abstract data type. It is a linear data structure. The principle of stack is LIFO (Last in First out).

- ✓ In Stack, elements are inserted or deleted only at one end. This is also called "top of the stack". An insert operation in stack is called push().
- ✓ A delete operation in stack is called pop().

Implementation (or) representation of Stack : A Stack can be implemented in two ways. They are:

- i) Stack using arrays
- ii) Stack using linked list

Operations of Stack ADT :

- a) Creation of Stack
- b) push()
- c) pop()
- d) Display

a. Creation of stack using arrays : Using arrays is a static process of creation of stack.

Algorithm:

1. Start
2. Read size, Create array S[size]
3. top = -1
4. Stop

b. Push() : Inserting an element into stack is known as push() operation. If the stack is full then it is known as 'Stack Overflow'.

Algorithm :

1. Start
2. Read n
3. If($\text{top} = (\text{size}-1)$) then
 Print "Stack Overflow"
else
 $\text{top} = \text{top} + 1$
 $S[\text{top}] = n$
4. Stop

c. Pop() : Deleting an element from the stack is known as pop() operation. If the stack is empty then it is known as 'Stack Underflow'.

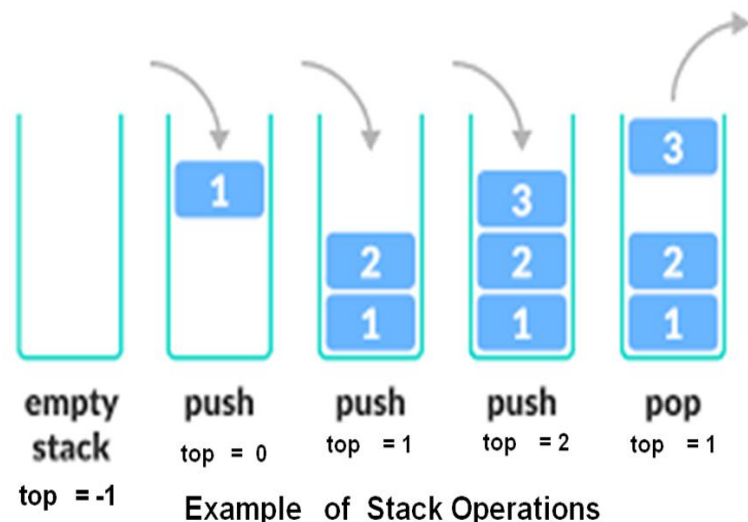
Algorithm :

1. Start
2. if ($\text{top} = -1$) then
 Print ' Stack Underflow'
else
 $\text{temp} = S[\text{top}]$
 "Deleted element" , temp
 $\text{top} = \text{top} - 1$
3. Stop

d. Display() : Display is the process of displaying (or) printing all elements.

Algorithm :

1. Start
2. for($i = \text{top}$; $i \geq 0$; $i--$)
 Print $S[i]$
3. Stop



2Q. Define Stack. Explain implementation of stack using linked list.**----- 10 Marks**

Ans: Stack ADT : A Stack is an abstract data type. It is a linear data structure. The principle of stack is LIFO (Last in First out).

- ✓ In Stack, elements are inserted or deleted only at one end. This is also called "top of the stack".
- ✓ An insert operation in stack is called push()
- ✓ A delete operation in stack is called pop()

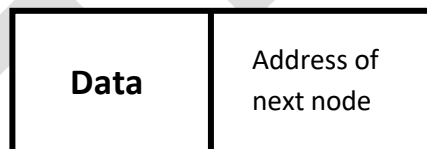
Implementation (or) representation of Stack : A Stack can be implemented in two ways. They are:

- i) Stack using arrays
- ii) Stack using linked list

Operations of Stack ADT :

- a) Creation of Stack
- b) push()
- c) pop()
- d) Display

a. Creation of stack using linked list : Using linked list is a dynamic process of creation of stack. Here, each element is also known as node.

Structure of node:**Algorithm:**

1. Start
2. Create node first
3. first=null
4. Stop

b.Push() : Inserting an element at the beginning of the list (Begin Insert) is known as push() operation.

Algorithm :

1. Start
2. Create node newnode
 - i. Read newnode.data
 - ii. newnode.next = null
3. if(first=null) then
first = newnode
else
newnode.next= first
first=newnode
4. Stop

c . Pop() :Deleting an element from the beginning of the list (Begin Delete()) is known as pop() operation.

Algorithm :

- 1.Start
2. Create node temp
3. first = temp.next
temp.next = null
4. Stop

d. Display() : Display is the process of printing all the elements.

Algorithm :

1. Start
2. Create node temp
3. for(temp=top;temp!=null;temp=temp.next)
Print temp.data
4. Stop

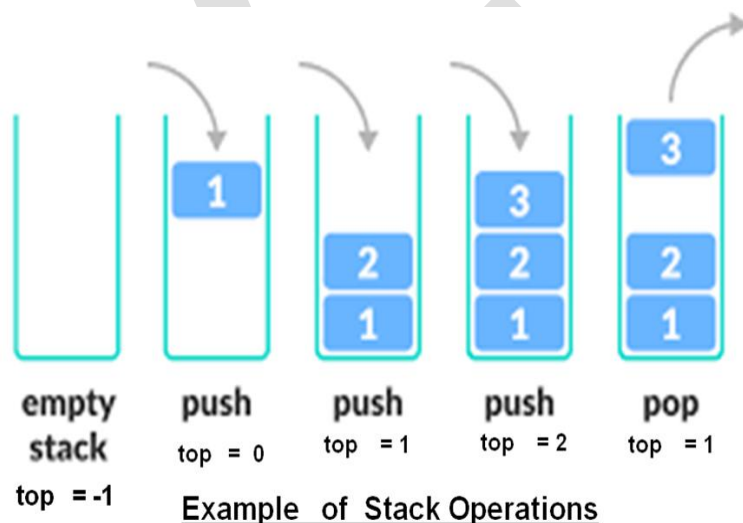
Q. Define a stack. Explain applications of stack.

Ans: Stack ADT : A Stack is an abstract data type. It is a linear data structure. The principle of stack is LIFO (Last in First out).

- ✓ In Stack, elements are inserted or deleted only at one end. This is also called "top of the stack".
- ✓ An insert operation in stack is called push()
- ✓ A delete operation in stack is called pop()

Implementation (or) representation of Stack : A Stack can be implemented in two ways. They are:

- i) Stack using arrays
- ii) Stack using linked list

Diagrammatic**Representation of Stack:****Applications of Stack:**

1. Stack is used in developing gaming applications.
2. Stacks are used in "undo operations" in M.S office, Photoshop.
3. Stacks are used in implementing function techniques, recursion techniques in programming language.
4. Stacks are used in calculating arithmetic expressions.
5. Stacks are used in implementing graph transversal algorithms DFS (Depth First Search) .
6. Stacks are used in developing programming language compilers.

Q. Define stack. Explain how to convert infix to prefix expression.

(Or)

Q. What is polish notation and reverse polish notation. Explain process to convert polish to reverse polish notation. --- 10 Marks (V.Imp)

Ans. A stack is an Abstract Data Type. It is a linear data structure.

→ The principal of stack is "LIFO".

→ Stacks are used in converting Infix to Postfix notation.

Types of Expressions: An expression is a collection of operators and operands. There are three types of notations for Arithmetic Expression in computer science. They are

1. Infix Expression (or) Notation
2. Prefix Expression (or) Notation
3. Postfix Expression (or) Notation

Infix Expression: In an expression, if an operator is placed between the operands then, it is known as Infix Notation. It is also known as "Polish Notation".

Ex: $a + b$

Here a, b are Operands

$+$ is Operator

$a + b$ is Infix notation

Prefix Expression: In an Expression, if an operator is placed before the operands then it is known as Prefix Expression.

Ex:- $+ab$

Here, a, b are operands

$+$ is Operator

$+ab$ is Prefix Notation

Postfix Expression: In an Expression if an operator is placed after operands then it is called Postfix Expression. It is also known as "Reverse polish Notation".

Ex: $ab+$

Here, a, b are operands

$+$ is Operator

$ab+$ is Postfix Expression

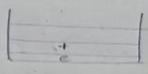
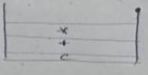
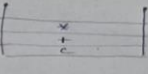
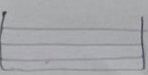
Converting Infix to Postfix Expression: Infix to postfix conversion is used in evaluating Mathematical Expression. We use stack data structure for converting Infix Notation to Postfix Notation

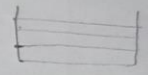
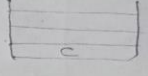
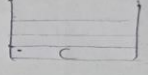
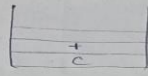
Algorithm :

1. Start
2. Read Infix expression or Notation
3. Add parenthesis "(" & ")" to the expression
4. Consider a Stack with max size
5. Moving from left to right ,read each symbol and do the following
 - i. If the symbol is "(", then push it into stack
 - ii. If the symbol is ")", then pop all the elements until "("
 - iii. If the symbol is "Operand" , then print it in output
 - iv. If the symbol is "operator", then push into stack. However, if stack contains operator with greater or equal priority then print it in output, then push new symbol(operator) into stack.
6. Stop.

Example for converting Infix to Postfix notation:(Or) Polish to reverse polish notation:

Consider Infix expression $(a+b*c)$

Infix Notation Input	Stack Operation Max-Size	Postfix Notation Output
5) b		ab
6) "x"		ab
7) c		abc
8) ")"		abc * +

Infix Notation Input	Stack Operation Max-Size	Postfix Notation Output
1) Start		
2) "("		
3) a		a
4) +		a

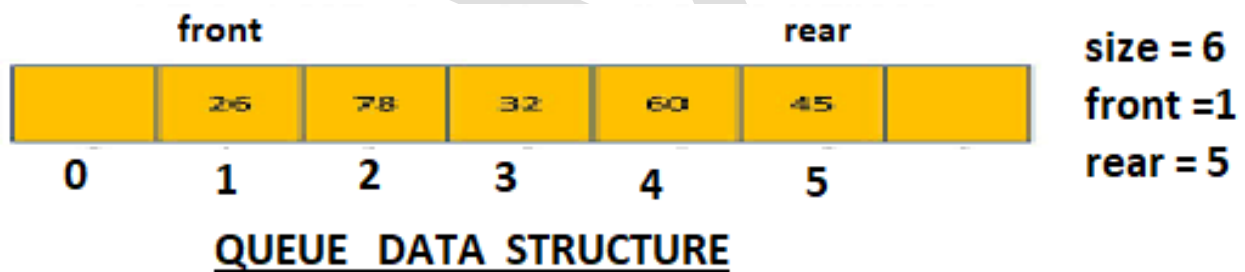
Therefore

- Infix Expression : $a+b*c$
- Postfix Expression : $abc * +$

Q. Define Queue. Explain Queue operations. (or)**Define Queue. Explain implementation of queue using arrays.****----- 10 Marks (Very Imp)**

Ans: **Queue ADT** : A Queue is an abstract data type. It is a linear data structure. The principle of queue is FIFO (First In First Out).

- A queue contains two ends. They are : i. Rear ii. Front
- An insert operation is called as Enqueue.
- An insert operation is done at rear when we insert, rear is incremented by one i.e; $\text{rear} = \text{rear} + 1$.
- A delete operation is called Dequeue.
- A delete operation is done at front, when we delete front is incremented by 1 i.e; $\text{front} = \text{front} + 1$.
- The default value of rear is -1 and front is 0.

Diagrammatic Representation of Queue:

Representation or implementation of queue :- A queue can be represented in two ways. They are :

- Using Arrays
- Using Linked list

Operations of Queue ADT :-

- a) Creation of queue
- b) Insert or Enqueue
- c) Delete or Dequeue
- d) Display

a) Creation of queue :- Using arrays, it is a static process of creation of queue.

Algorithm :

1. Start
2. Read Size, Create array Q[size]
3. rear = -1, front = 0
4. Stop

b) Insert or Enqueue(): Inserting an element into the queue is known as enqueue(). When queue is full it is known as 'Queue Overflow'.

Algorithm :

1. Start
2. Read n
3. if (rear = size-1) then
 Print 'Queue Overflow'
- else
 rear=rear+1
- Q[rear]=n
4. Stop

c) Delete or Dequeue () :Deleting an element from the queue is known as dequeue(). If the queue is empty then it is called as "queue underflow".

Algorithm :-

1. Start
2. if(front = rear+1) then
 Print 'queue underflow'
- rear = -1
- front = 0
- else
 Print "deleted element is" Q[front]
- front=front+1
3. Stop

d) Display() : Display is the process of printing all the elements in a queue

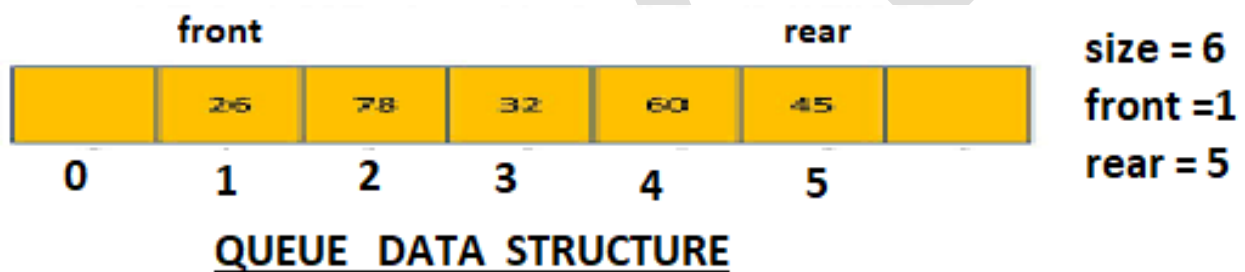
Algorithm :-

1. Start
2. for(i=front;i<=rear;i++)
 Print Q[i]
3. Stop

Q. Define Queue. Explain implementation of queue using linked list.

Ans: Queue ADT : A Queue is an abstract data type. It is a linear data structure. The principle of queue is FIFO (First In First Out).

- A queue contains two ends. They are : i. Rear ii. Front
- An insert operation is called as Enqueue.
- An insert operation is done at rear when we insert, rear is incremented by one i.e; $\text{rear} = \text{rear} + 1$.
- A delete operation is called Dequeue.
- A delete operation is done at front, when we delete front is incremented by 1 i.e; $\text{front} = \text{front} + 1$.
- The default value of rear is -1 and front is 0.

Diagrammatic Representation of Queue:

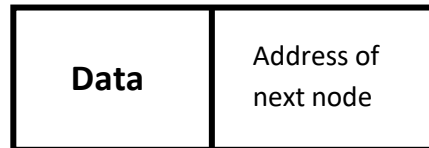
Representation or implementation of queue :- A queue can be represented in two ways. They are :

- Using Arrays
- Using Linked list

Operations of Queue ADT :-

- e) Creation of queue
- f) Insert or Enqueue
- g) Delete or Dequeue
- h) Display

Creation of queue :- Creation of queue using linked list is a dynamic process where each element is called node.

Structure of node:**Algorithm:**

1. Start
2. Create node first,last
 first = null
 last = null
3. Stop

Enqueue(): Inserting an element into the queue(EndInsert()) is known as enqueue().

Algorithm :

1. Start
2. Create newnode
 - i. read newnode.data
 - ii. newnode.next = null
3. if (first=null) then
 first = newnode
 last = newnode
 else
 last.next=newnode
 last=newnode
4. Stop

Dequeue(): Deleting an element from the queue(begindelete()) is known as dequeue().

Algorithm :

1. Start
2. Create node temp
3. temp = first
 first = first.next
 temp.next = null
4. Stop

Display() : Display is the process of printing all the elements in a queue

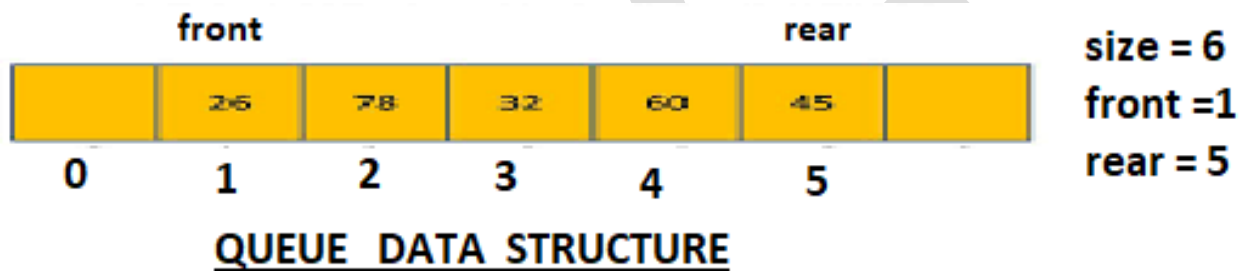
Algorithm :

1. Start
2. Create node temp
3. for(temp = first; temp!=null; temp=temp.next)
 print temp.data
4. Stop

Q. Define queue. Explain applications of queue.

Ans: Queue ADT : A Queue is an abstract data type. It is a linear data structure. The principle of queue is FIFO (First In First Out).

- A queue contains two ends. They are : i. Rear ii. Front
- An insert operation is called as Enqueue.
- An insert operation is done at rear when we insert, rear is incremented by one i.e; $\text{rear} = \text{rear} + 1$.
- A delete operation is called Dequeue.
- A delete operation is done at front, when we delete front is incremented by 1 i.e; $\text{front} = \text{front} + 1$.
- The default value of rear is -1 and front is 0.

Diagrammatic Representation of Queue:

Representation or implementation of queue :- A queue can be represented in two ways. They are :

- Using Arrays
- Using Linked list

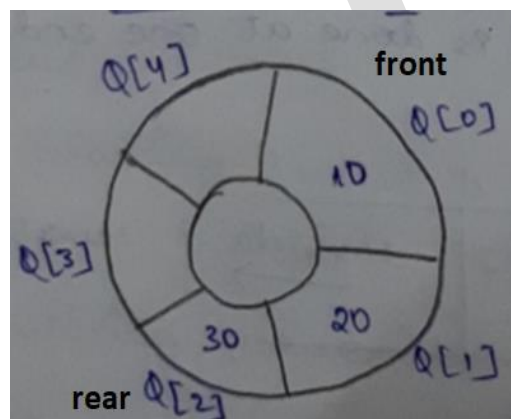
Applications of Queue:

1. Queues are used in CPU Scheduling
2. Queues are used in memory management
3. Queues are used in file management
4. Queues are used in disc management
5. Queues are used in downloading files from internet.
6. Queues are used in data transfer in networking & internet.
7. Queues are used in implementing graph traversal algorithm "Breadth First Search"(BFS).

Q. Define Queue. Explain about circular queue.**----- 5Marks (Imp)**

Ans: Circular Queue ADT : A Circular Queue is an abstract data type. It is a linear data structure. In Circular queue, "last element is connected to the first element in a circular manner".

- A Circular queue has two ends, they are: 1. Rear 2. Front.
- Default value of rear is -1 and front is 0.
- An insert operation is called as Enqueue.
- An insert operation is done at rear , when we insert, rear is incremented by one i.e ; **$\text{rear} = (\text{rear} + 1) \% \text{size}$** .
- A delete operation is called Dequeue.
- A delete operation is done at front, when we delete front is incremented by 1 i.e; **$\text{front} = (\text{front} + 1) \% \text{size}$** .

Diagrammatic Representation of Circular Queue:

Representation or implementation of Circular Queue ADT:- A Circular queue can be represented in two ways. They are :

- Using Arrays
- Using Linked list

Operations of Circular Queue ADT :-

- Creation of queue
- Insertion or Enqueue
- Deletion or Dequeue
- Display

Creation of Circular Queue :- It is a static process of creation of queue

Algorithm :-

Start

Read size, Create array Q[size]

Rear = -1

Front = 0

Stop

Enqueue () :- Inserting an element into the queue is known as enqueue(). When queue is full it is known as 'Queue Overflow'.

Algorithm :-

1. Start
2. Read n
3. If((rear=(size-1) && front ==0)|| (front= rear+1))
Print 'Queue Overflow'
- else
 rear=(rear+1)%size
 Q[rear]=n
4. Stop

Dequeue () : Deleting an element from the queue is known as dequeue(). If the queue is empty then it is called as "queue underflow".

Algorithm :-

1. Start
2. if((front =rear+1) then
 Print 'queue underflow'
 Rear=-1, front = 0
- else
 front=front+1
 Print "deleted element" Q[front]
1. Stop

Display() : Display is the process of printing all the elements in a queue

Algorithm :-

1. Start
2. for(i=front;i<=rear;i++)
 Print Q[i]
3. Stop

Q. Define Queue. Explain types of queues.**----- 5Marks(Imp)**

Ans: Queue ADT : A Queue is an abstract data type. It is a linear data structure. The principle of queue is FIFO (First In First Out).

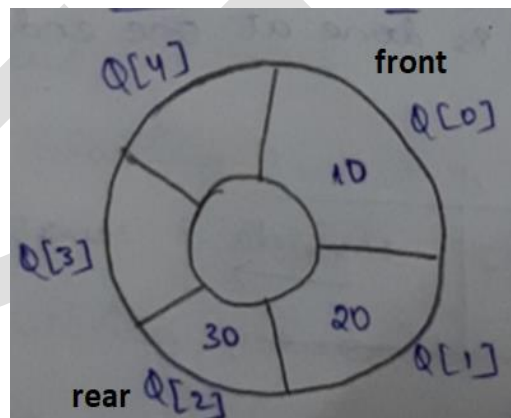
- A queue contains two ends. They are : i. Rear ii. Front
- An insert operation is called as Enqueue.
- A delete operation is called Dequeue.

Types of Queues:

1. Circular Queue
2. Priority queue
3. Double – Ended Queue

Circular Queue ADT : A Circular Queue is an abstract data type. It is a linear data structure. In Circular queue, "*last element is connected to the first element in a circular manner*".

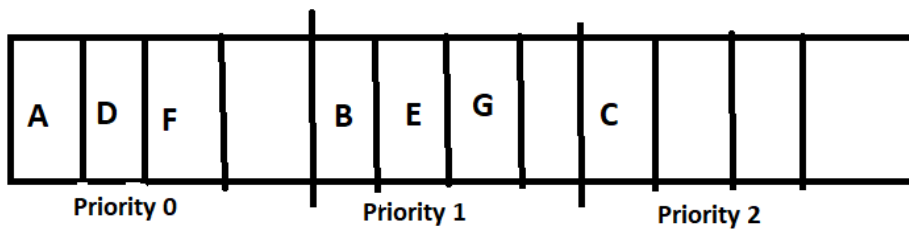
- A Circular queue has two ends, they are: 1.Rear 2. Front.
- Default value of rear is -1 and front is 0.
- An insert operation is called as Enqueue.
- An insert operation is done at rear , when we insert, rear is incremented by one i.e ; **$\text{rear} = (\text{rear} + 1) \% \text{size}$** .
- A delete operation is called Dequeue.
- A delete operation is done at front, when we delete front is incremented by 1 i.e; **$\text{front} = (\text{front} + 1) \% \text{size}$** .

Diagrammatic Representation of Circular Queue:

Priority Queue: A priority queue is a linear data structure, it follows "FIFO" (First In First Out) principle.

- In Priority queue , a queue is divided into no,of queues.
- Elements are inserted and deleted into queue , based on the priority NUMBER..
- Priority Queue is used in operating system for multi-processing

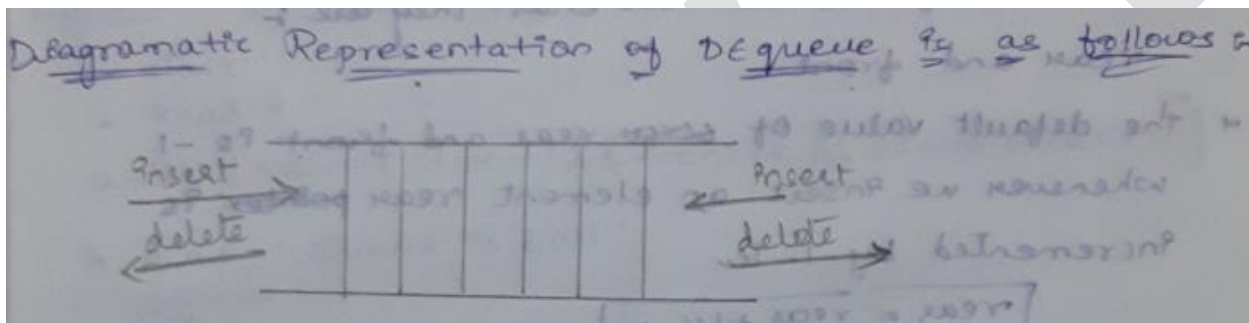
Diagrammatic Representation of Priority Queue:



Element	priority
A	0
B	1
C	2
D	0
E	1
F	0
G	1

Deque: Dequeue stands for "Double-Ended Queue".

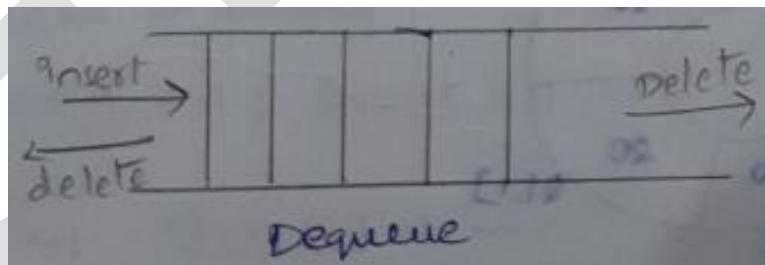
→ In dequeue, we can insert and delete elements from both ends.



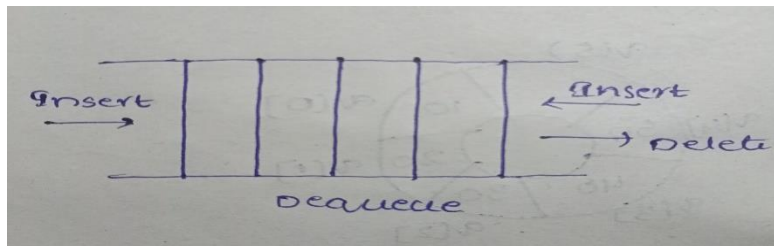
Types of Dequeues:

1. Input Restricted Dequeue
2. Output Restricted Dequeue

Input Restricted Dequeue: Here, Insertion is done at one end, but deletion is done at both ends.



Output Restricted Dequeue: Here, Deletion is done at one end, but insertion is done at both ends.



//Program To Implement a Stack Using Array

```

import java.util.*;
class stack
{
    public int size;
    int stackarray[];
    int top;
    public stack(int s)
    {
        size = s;
        stackarray = new int[size];
        top = -1;
    }
    void push(int n)
    {
        if(top== (size-1))
        {
            System.out.println("\tStack Over flow");
        }
        else
        {
            top = top+1;
            stackarray[top] = n;
        }
    }
    void pop()
    {
        if( top == -1)
            System.out.println(" \tStack Under Flow");
        else
        {
            // temp= stackarray[top];
            // System.out.println("\tPop Element is"+temp);
            top = top -1;
        }
    }
    void display()
    {
        int i;
        for(i=top;i!=-1;i--)
            System.out.println("\t"+"|"+stackarray[i]+"|");
    }
}

```

```

class stackarray
{
public static void main(String args[])
{
stack s=new stack(5);
int n,ch;
Scanner in=new Scanner(System.in);
try
{
do
{
System.out.println("\nEnter ur choice");
System.out.println("\t1.Push\t2.Pop\t3.Display\t4.Stop");
ch= in.nextInt();
switch(ch)
{
case 1: System.out.println("Enter an element");
n = in.nextInt();
s.push(n);
break;
case 2:
s.pop();
break;
case 3:
s.display();
break;
}
System.out.println("\nEnter ur choice");
}while(ch<4);
}catch(Exception e)
}
}

```

//Program To Implement a Stack Using Linked List

```

import java.util.*;
class node
{
    public int data;
    public node next;
    node(int n)//10
    {
        data = n; //10
        next = null;
    }
}
class stackll
{
    node first;
    stackll()
    {
        first = null;
    }
    void push(int x) //10
    {
        node newnode=new node(x); //10
        if( first == null)
        {
            first = newnode;
        }
        else
        {
            newnode.next = first;
            first = newnode;
        }
    }
    void pop() //begindelete()
    {
        System.out.print("Deleted element is "+first.data+"\n");
        node temp; temp=first; first=temp.next; temp.next=null;
    }
    void display()
    {
        node temp;
        for(temp=first;temp!=null;temp=temp.next)
        System.out.print("\t"+temp.data);
        System.out.println();
    }
}
} // class stacklinkedlist finished

```

```

class stacklinkedlist
{
    public static void main(String args[])
    {
        stackll s=new stackll();
        int ch,n;
        Scanner in = new Scanner(System.in);

        try
        {
            do
            {
                // System.out.println("\tEnter ur choice");
                System.out.println("\t1.Push\t2.Pop\t3.Display\t4.Stop");
                ch= in.nextInt();
                switch(ch)
                {
                    case 1: System.out.println("Enter an element");
                        n = in.nextInt();
                        s.push(n);
                        break;
                    case 2:
                        s.pop();
                        break;
                    case 3:
                        s.display();
                        break;
                }
                System.out.println("\tEnter ur choice");
            }while(ch<4);
        }catch(Exception e){}

    }
}

```

//Program To Implement a Queue Using Array

```

import java.util.*;
class queue
{
int maxsize;
public int[] queuearray ;
int front,rear;
queue(int x)
{
maxsize=x;
queuearray=new int[maxsize];
front = 0;
rear = -1;
}
void insert(int n)
{
if(rear == maxsize -1)
System.out.println("Queue is full");
else
{
rear= rear+1;
queuearray[rear] = n;
}
}
void delete()
{
if (front == rear +1)
{
System.out.println("Queue is empty");
rear = -1;
front = 0;
}
else
{
System.out.println("Deleted Element is"+queuearray[front]);
front = front + 1;
}
}
void display()
{
for(int i=front;i<=rear;i++)
System.out.print(queuearray[i]+" ");
}
}

```

```
class queuearray
{
public static void main(String args[])
{
queue q=new queue(5); //constructor
Scanner s=new Scanner(System.in);
int n;
int ch;
try
{
do
{
System.out.println("\n1. Insert 2.Delete 3. Display 4. Exit\n Enter your Choice");
ch=s.nextInt();
switch(ch)
{
case 1: System.out.println("Enter value");
n=s.nextInt();
q.insert(n);
break;
case 2: q. delete();
break;
case 3: q.display();
break;
}}while(ch<4);
}catch(Exception e){}
}
}
```

//Program To Implement a Queue Using Linked List

```

import java.util.*;
class node
{
    public int data;
    public node next; public node(int x)
    {
        data=x;
        next=null;
    }
}
class queuesll
{
    node first,last;
    queuesll()
    {
        first=null;
        last=null;
    }
    void enqueue(int n)
    {
        node newnode=new node(n);
        if(first==null)
        {
            first=newnode; last=newnode;
        }
        else
        {
            last.next=newnode; last=newnode;
        }
    }
    void dequeue()
    {
        System.out.print("enqueue element is "+first.data+"\n");
        node temp; temp=first; first=temp.next; temp.next=null;
    }
    void display()
    {
        node temp; for(temp=first;temp!=null;temp=temp.next)
        System.out.print("\t"+temp.data);
        System.out.println();
    }
}

```

```
class queueLinkedList
{
    public static void main(String args[])
    {
        queueLL s=new queueLL();
        int ch,n;
        Scanner sc=new Scanner( System.in); try
        {
            do
            {
                System.out.println("Enter your choice\n 1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Exit" );
                ch=sc.nextInt();
                switch(ch)
                {
                    case 1: System.out.println("Enter a number");
                        n=sc.nextInt(); s.enqueue(n);
                        break;
                    case 2:s.dequeue(); break;
                    case 3:s.display(); break;
                }
            }
            while(ch<4);
        }
        catch(Exception e)
        {
        }
    }
}
```