Data Structure
Chapter 2: stacks, queues

GATE CS Lectures

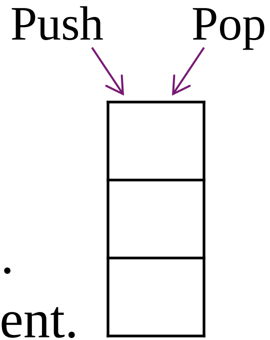
By

Monalisa Pradhan

- **Section 4: Programming and Data Structures**
  Programming in C. Recursion.Arrays, stacks, queues, linked lists, trees, binary search trees, binary heaps, graphs.

- Chapter 1:Arrays
- Chapter 2: stacks, (Stack permutation , Postfix , Recursion, TOH) queues (Linear Queue, Circular Queue)
- Chapter 3: linked lists
- Chapter 4: trees, binary search trees, binary heaps
- Chapter 5: graphs

- **Stack**
- LIFO or FILO model
- One side open the other side is closed.
- Operation :
- PUSH(X)- Check overflow condition and Insert an element X.
- POP()- Check Underflow condition then delete top most element.
- Some more operation are PEEK(),CHANGE(),ISEMPTY(),ISFULL(),GET TOP()
- Stack Permutation :
- Each element Pushed in/Pop out.
- The elements are poped out based on design sequence.



- No of Stack Permutation = $\frac{{}^{2n}C_n}{n+1}$

N	**2**	**3**	**4**
Stack permutation	2(12,21)	5(123,132,213,231,321)	14

Precedence	**Associatively**
(),{},[]	
^	Right-Left
*,/	Left-Right
+,-	Left-Right

Infix expression evaluation

(1+2)*3-10/5+2^3^1

=3*3-10/5+2^3^1

=3*3-10/5+2^3

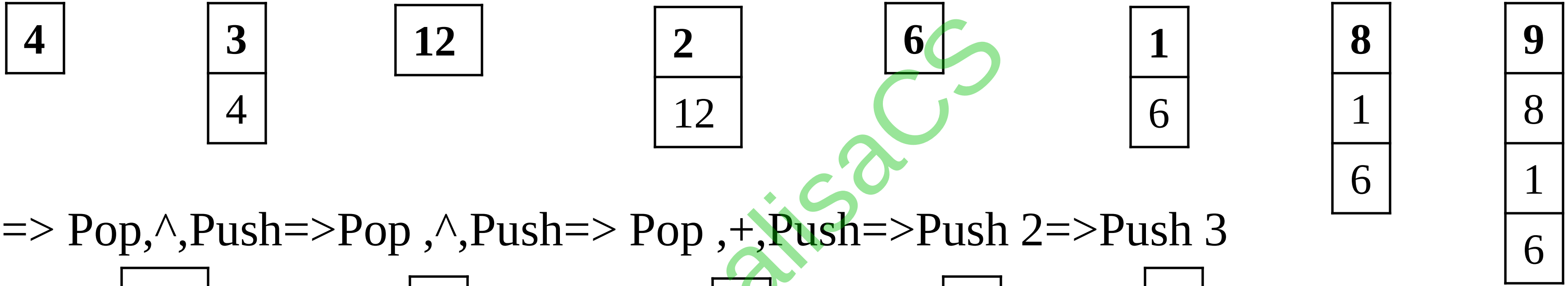
=3*3-10/5+8

=9-10/5+8

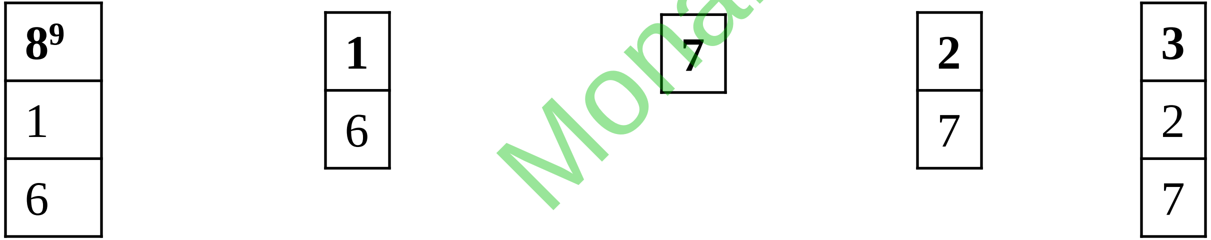
=9-2+8 = 7+8 =15

- **Infix** :<operand1><operator><operand2>
- **Prefix** : <operator> <operand1>< operand2>
- **Postfix**:<operand1><operand2><operator>
- Example : Infix : (A+B) * (C-D) , Prefix : *+AB-CD , Postfix: AB+CD-*
- Conversion Process :The relative position of operand not changed. Position of operator change as per precedence and associative rule .
- Infix: (1+2)*3-10/5+2^3^1
- Postfix: 1 2+ 3* 10 5/- 2 3 1^^+
- Prefix: +-*+1 2 3 /10 5 ^2 ^3 1

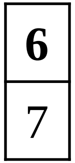
- **How to evaluate postfix expression using stack**
- 1.Operand :Push 2.Operator :Pop top 2 operand ,evaluate, Push result
- Postfix : 4 3 * 2 / 1 8 9^ ^ +2 3 * - [Infix :4*3/2+1^8^9 -2*3]
- Push 4=>Push 3 =>Pop,*,Push=>Push 2 =>Pop ,/,Push=>Push 1=>Push 8=>Push 9



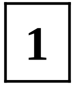
- => Pop,^,Push=>Pop ,^,Push=> Pop ,+,Push=>Push 2=>Push 3



- => Pop,*,Push



- => Pop,-,Push



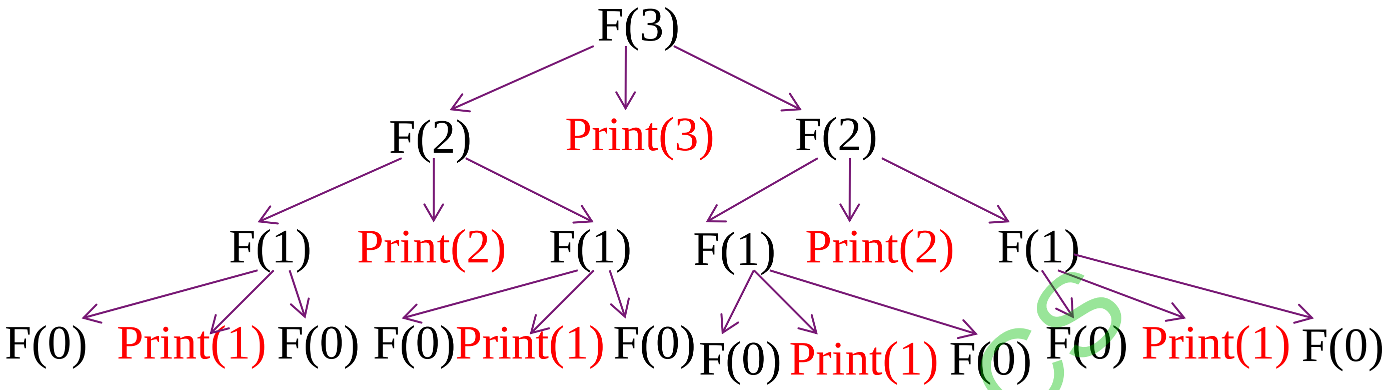
- **Recursion:**
- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.

Types of Recursion

Direct		Indirect	Nested	Excessive

Tail
Ex:Fact
	Non Tail
	Ex: ToH
		A() {B()}
		B() {A()}
			A(){B()}
			B(){C()}…
				Ex:Fibonacci no

- **Tail Recursion**: A recursive function is tail recursive when recursive call is the last thing executed by the function.
- **Excessive Recursion** : the amount of stack space required increases dramatically with the amount of recursion that occurs.
- This can lead to program crashes if the stack runs out of memory.
- It doesn’t remember previous evaluated value .Recursion by default excessive .
- Stack contain inactive record .
- In Recursion tree the behavior of tracing is preorder .


F(3)
F(2) Print(3) F(2)
F(1) Print(2) F(1) F(1) Print(2) F(1)
F(0) Print(1) F(0) F(0) Print(1) F(0) F(0) Print(1) F(0) F(0) Print(1) F(0)

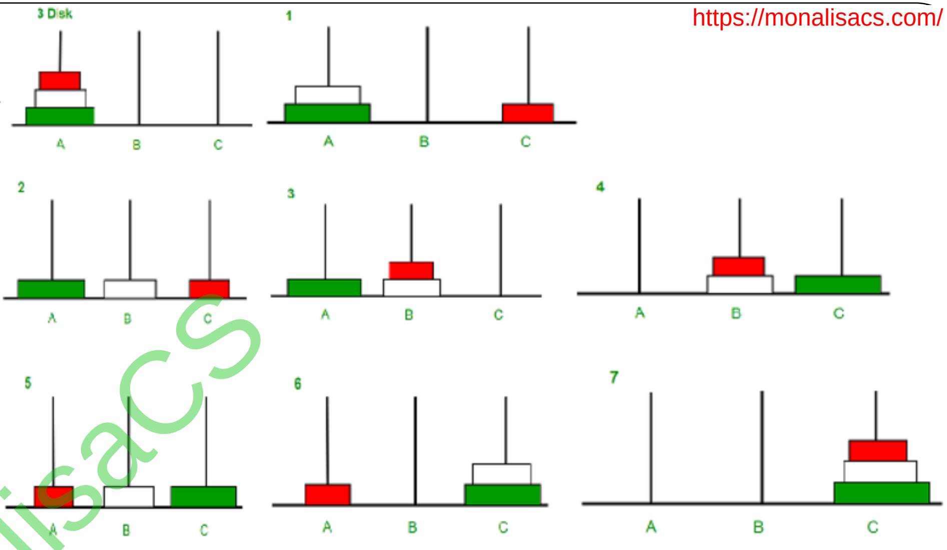
Ex:

```
F(x)
{
F(x-1)
Print (x)
F(x-1)
}
```

Output for F(3) ?

- Output for F(3) is 1213121
- **Tower of Hanoi**
- Tower of Hanoi is a mathematical puzzle where we have three rods and n disks.
- The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
- 1) Only one disk can be moved at a time.
- 2) a disk can only be moved if it is the uppermost disk on a stack.
- 3) No disk may be placed on top of a smaller disk.
- Take an example for 2 disks :
- Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.

- Step 1 : Shift first disk from 'A' to 'B'.
- Step 2 : Shift second disk from 'A' to 'C'.
- Step 3 : Shift first disk from 'B' to 'C'.
- The pattern here is :
- Shift 'n-1' disks from 'A' to 'B'.
- Shift last disk from 'A' to 'C'.
- Shift 'n-1' disks from 'B' to 'C'.

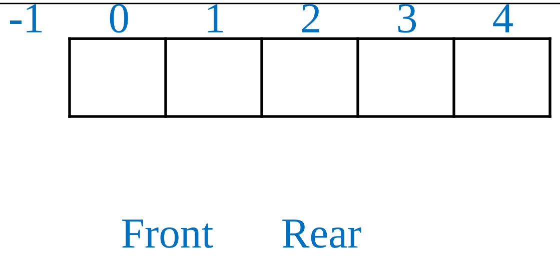


```
Void TOH(int n,char L,char M,Char R)
{If n!=0
TOH(n-1,L,R,M)
Print L to R
TOH(n-1,M,L,R)
}
```

- Recurrence relation of TOH :$T(n)=2T(n-1)+1$
- Total move=2^n -1

n	**1**	**2**	**3**	**4**
No of move	1	3	7	15

- **Queue**
- FIFO OR LILO model.
- Pointers: Front,Rear.
- Operation:
- **Enqueue:** Inserts an item to the queue.
- **Dequeue:** Deletes an item from the queue.
- Some other operation are **peek(),isfull() ,isempty()**
- In a **linear queue**, the traversal through the **queue** is possible only once,once an element is deleted, we cannot insert another element in its position.

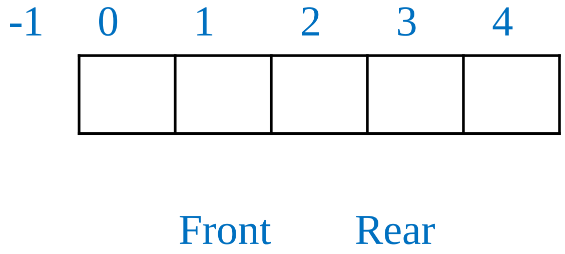


- Ex: enqueue(2), enqueue(5), enqueue(7),dequeue, enqueue(6), enqueue(1),dequeue, enqueue(8),

```
Void enqueue(int x)
{ If (rear==N-1)
    print (“overflow”);
else if (front==-1 && rear==-1)
  {  front=rear =0;
    queue[rear]=x;}
else {rear++;
queue[rear]=x;}
```

```
void dequeue()
{if (front==-1 && rear==-1)
   print (“Underflow”)
 else if(front =rear)
   front=rear=-1;
 else
    front++
}
```

- This **disadvantage** of a **linear queue** is overcome by a circular **queue.**
- **Circular Queue:**
- In Circular Queue (FIFO) the last position is connected back to the first position of Queue to make a circle
- Operations on Circular Queue: enqueue,dequeue .



```
Void enqueue(int x)
{if ((rear+1)%N)==front
    print(overflow);
 else if(front==-1&& rear==-1)
    {front=rear=0;
    queue[rear]=x;}
 else
   { rear =(rear+1)%N;
     queue[rear]=x;} }
```

```
void dequeue()
{
if (front==-1 && rear==-1)
   print (“Underflow”)
else if(front =rear)
  front=rear=-1;
 else
   front=(front+1)%N
}
```

- Ex: enqueue(2), enqueue(5), enqueue(7),dequeue, enqueue(6), enqueue(1),dequeue, enqueue(8), enqueue(4), enqueue(0), dequeue,