Module: 2

STACK

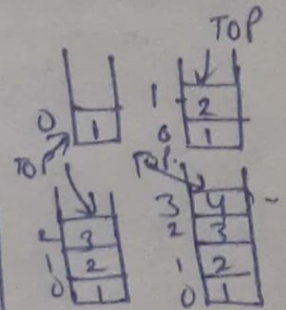
Stack: It is a linear data structure in which insertion or deletion of elements takes place at the same end. i.e, element is inserted or removed from one end only & that end is called 'TOP' of stack.

Ex- Plates in a cafeteria.

Operations: PUSH & POP.
(insert) (Remove)

Last In First Out
LIFO

Ex- 1,2,3,4



PUSH.

When removing elements '4' is removed at first, than 3 & so on.

~~Stacks ADT~~:

PUSH.

```
push(stack, data)
if stack is full
return overflow.
end if.
  top ← top+1.
  Stack[top] ← data.
end.
```

POP

```
pop(stack).
if stack is empty.
  return underflow.
end if.
data ← stack[top]
  top ← top-1.
  return data
  end.
```

STACK ADT: It is an abstract data type that serves as a collection of elements, with two basic operations: push and pop.

Scanned by CamScanner

Types of Notations:

1. **Infix**: operators are placed in between operands.
   Ex- A+B*C. // A, B, C → operands
   *, + - operators
2. **Prefix**: when operators are placed before the operands, it is called prefix.
   +A*BC.
3. **Postfix**: Operators are placed after operands.
   ABC*+.

* Evaluation of Postfix:

Steps: -

1. Scan the array symbols one by one from left to right.
2. if symbol is operand, push into stack.
3. symbol is operator then, pop last two element of stack & solve them push it to stack.
4. do the same until array do not end.
5. Pop the elements from stack,

Scanned by CamScanner

Ex - 456*+ .

Step	I/p symbol	operation	stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6.	
4.	*	Pop (2 elements) & evaluate.	4	5*6
5.	Nil	Push result to stack	4,30	
6.	+	Pop (2 elements) & evaluate.	empty.	4+30 =34
7.		Push result (34)	34	
8.	-	No-more element -pop	empty	(34) result

Ex 4, 5, 4, 2, ^, +, *, 2, 2, ^, 9, 3, /, *, - .

Result : 72.

Conversion of Infix to Prefix & Post fix Notation:

Infix To Postfix Conversion:

Algo -

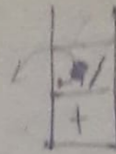
1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else
   - if precedence (priority) of scanned operator is greater than the precedence of operator in stack, push it.
   - Else pop the operator from the stack until the precedence of the scanned operator is less-equal to the precedence of the operator residing on the top of stack. Push the scanned operator to the stack.
4. If the scanned character is an '(' push into stack
5. If scanned character is an ')' pop and output from the stack until an '(' is encountered.
6. Repeat step 2-6 until infix expression is scanned.
7. Pop and output from the stack until it is not empty.

Scanned by CamScanner

Ex 3+4*5/6

stack: + *

O/P - 345*



Precedence

high to low

^

*, /

+, -

I/p symbol	stack	output
3	NULL	3
+	+	3
4	+	34
*	+*	34
5	+*	345
/	+/	345*
6	+/	345*6
NULL	+	345*6/
NULL	NULL	345*6/+

Ex. (300+23)* (43-21) /(84+7)

I/p	stack	output
(	(	-
300	(	300
+	(+	300 23
23	(+	300 23
)	-	300 23+
*	*	300 23+
(	*(	300 23+
43	*(	300 23+43
-	*(-	300 23+43
21	*(-	300 23+43 21
)	*	300 23+43 21-

I/p	stack	out
/	/	300 23+43 21 -*
(	/(	300 23+43 21 -*
84	/(	300 23+43 21-* 84
+	/(+	300 23+43 21-*84
7	/(+	300 23+43 21-*84 7
)	/(	300 23+43 21-*84 7+
-	-	300 23+43 21-*84 7+/

Scanned by CamScanner

Infix To Prefix :

Eg (A+B^C)*D+E^5

Step1 Reverse the expression.

5^E+D*)C^B+A(

(2) Make every ')' as '(' and '(' as ')'

5^E+D*(C^B+A)

(3) Convert to postfix -

~~A5+EB^C~~.

I/p symbol	stack	output
5	-	5
^	^	5
E	^	5E
+	~~^~~+	5E^
D	+	5E^D
*	+*	5E^D
(	+*(	5E^D
C	+*(	5E^DC
^	+*(^	5E^DC
B	+*(^	5E^DCB
+	+*(+	5E^DCB^
A	+*(+	5E^DCB^A
)	+*	5E^DCB^A+
end	empty	5E^DCB^A+*+

Step4 Reverse the result

+*+A^BCD^E5.

Scanned by CamScanner

Recursion :

It is a process of expressing a function in terms of itself. When a function calls itself with reduced input & has a base condition to stop the process, i.e, in order to solve a problem recursively, it should follow two steps-

→ it should be in recursive form.

→ problem statement must include a termination condition.

Ex. **Calculation of factorial:**

```
#include <stdio.h>
void main()
{ int n;
long int fact(int n);
printf("enter the no=");
scanf("%d",&n);
printf("fact=%d",fact(n));
getch();
}
```

```
long int fact (int n)
{
  if (n<=1)
    return(1);
  else
  return (n * fact(n-1));
}
```

Prog of fibonacci series

```
int a[10];
  a[0]=0; a[1]=1
  for (i=2 ; i<10; i++)
  {
    a[i]=a[i-1]+a[i-2];
  }
```

0 1 1 2 3

0	1	2	3	4	5
0	1	1	2	3	5

Scanned by CamScanner

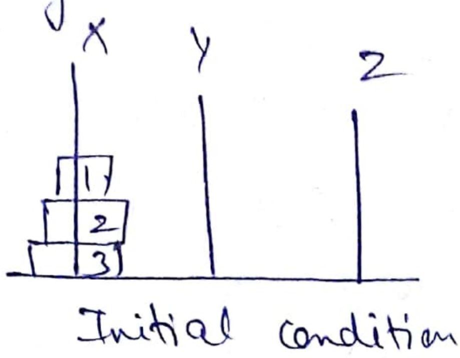
```
#include <stdio.h>
int fibo(int);
int main()
{ int num;
  int result;
pf("Enter nth no. in fib.
     series");
sf("%d", &num);
  if (num<0)
{
p.f("fib of negative no
is not possible");
}
else.
{ result = fibo(num);
pf("%d", result);
}. return0;
}
```

```
int fibo (int num)
{ if (num ==0)
  { return 0;
  }
  else if (num==1)
  { return1;
  }
else
{
 return (fibo(num-1)+
                fibo(num-2));
}
}.
```

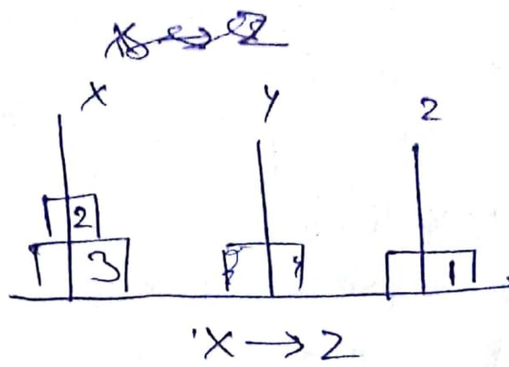
Tower of Hanoi: It is a game problem. There will be three diff size disks. At the beginning of game the disks are ~~attached~~ stacked on x peg, with largest on bottom and smallest on top. for n=3 disks

Rules:
- → one disk can move at a time.
- → larger disk can't placed on smaller one.
- → No. of steps $= 2^n - 1$. | Ex - $n = 3$. $2^3 - 1 = 7$

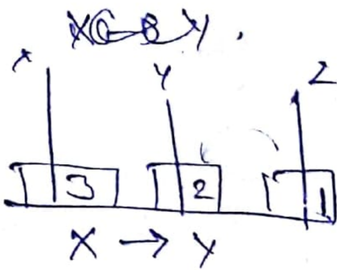
Ex for n = 3 disks.



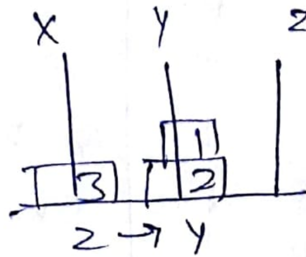
Initial condition



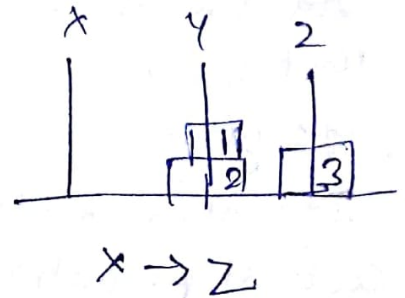
X → Z



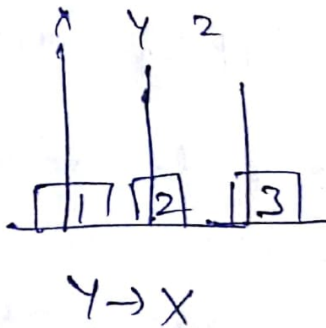
X → Y



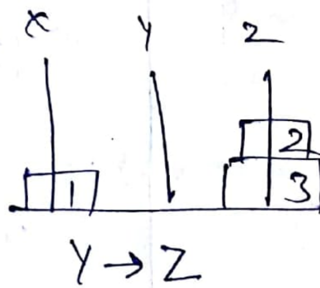
Z → Y



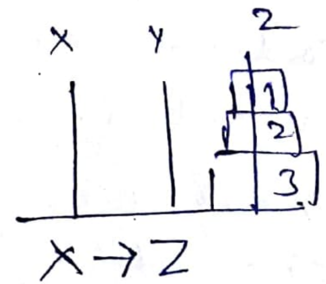
X → Z



Y → X



Y → Z



X → Z

Scanned by CamScanner

→ Recursion Vs Iteration:

Recursion:
- It require stack implementation.

Iteration	Recursion
1. It is a process of executing a statement or a set of statements repeatedly, until some condition is specified.	1. It is the technique of defining anything in terms of itself.
2. Iteration involves four clear cut steps initialization, condition, execution and updation.	2. There must be an exclusive if statement inside the recursive function specifying stopping condition.
3. Any recursive problem can be solved iteratively.	3. All problems do not have recursive solution.
4. It is more efficient in terms of memory utilization and execution speed.	4. It is a worse option for simple problems or problems not recursive in nature.

Scanned by CamScanner