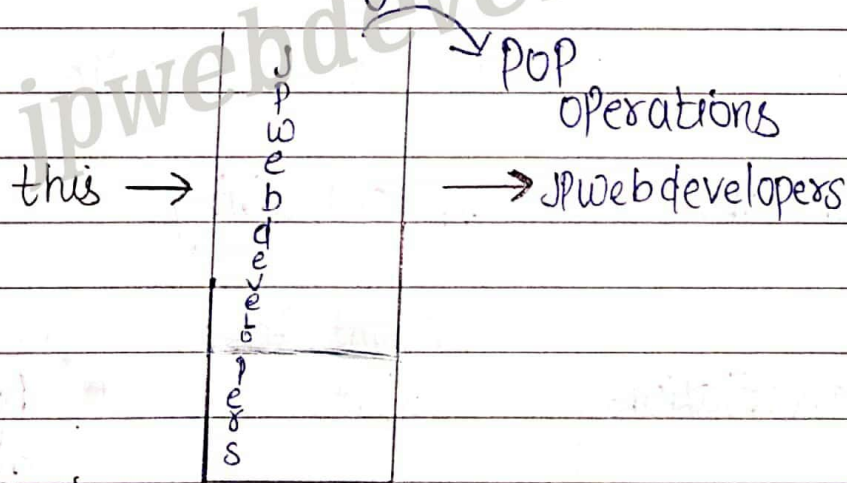


## \* Stack Applications :-

Stack are found many applications in Computer Science. Some of applications are as follows:-

### (i) String Reversal (Reverse a String) :-

A Simple stack algorithm could be used to reverse a word: push all the characters on the stack, then POP from the stack until it's empty.



### Algorithm :-

1. if Top = 0  
Print "UNDERFLOW" and RETURN
2. else
3. push all characters one by one.
4. Then pop all characters one by one.
5. finally, convert to the string.

## (2) Evaluation of Arithmetic Expressions:-

- An Arithmetic expression consists of operands and operators.
- In addition to operands and operators, the arithmetic expression may also include parenthesis like "left parenthesis" and "right parenthesis".

Example :-  $A + (B - C)$

- Precedence Rules for Arithmetic Expression:-

| Operators                  | Associativity | Precedence   |
|----------------------------|---------------|--|
| $\wedge$<br>exponentiation | Right to left | Highest followed by $*$ Multiplication and $/$ division. |
| $*$ $/$                    | Left to Right | Highest followed by $+$ , $-$                            |
| $+$ $-$                    | Left to Right | Lowest   |

Evaluation of Arithmetic Expressions requires two steps:-

ci) first, convert the given expressions into special notation.

cii) Evaluate the expressions in this new notation.

### Notations for Arithmetic Expressions:-

ci) Infix Notation.

cii) Prefix Notation.

ciii) Postfix Notation.

### Infix Notation:-

The operator symbol is placed between the two operands.

$$A+B$$

$$C*D$$

$$E/F$$

$$G-H$$

Example:  $A+B * C - D / F$

We can also use the concepts of round brackets along with the operands and operator to specify what operation is to be performed first along with the precedence level.

$$(A+B) * (C-D)$$

### Prefix notation (Polish Notation):-

It refers to the notation in which the operator

is placed before its two operands.

$$+AB \quad *CD$$

We can easily convert the infix notation to the Polish Notation, step by step, using the concept of square brackets [ ].

$$(A+B) * (C-D) = [+AB] * [-CD] \\ = *+AB-CD$$

### Reverse Polish Notation (RPN)

→ In this type of expression, operators comes after two operands.

$$AB+ \quad CD *$$

→ It is also called Postfix Notation.

→ We can easily convert the infix Notation to the Postfix Notation, step by step using the concept of square bracket [ ].

$$(A+B) * (C-D) \\ = [AB+] * [CD-] \\ = AB+CD *$$

## \* Evaluation of Postfix Expression:-

When an arithmetic expression is presented in the postfix form, you can use a stack to evaluate it to get value.

→ find the first operator from left to right and perform the operation.

→ It is free from any precedence.

Example:-  $a b c * -$

Here first operator is  $*$  therefore  $b$  and  $c$  is multiplied first.

$(b * c)$

Subtract  $a$  and multiplied operand to get in final result.

Example:-  $4 6 8 * -$

Step 1:-  $(6 * 8)$

Step 2:-  $4 48 -$

Step 3:-  $48 - 4$

Expression becomes  $44$

## Algorithm:-

Start reading elements from left to right.

1. Read the next element.
2. If the element is operand then:-  
push element in stack.
3. If element is operator - then
  - (a) pop two operands from stack
  - (b) Evaluate (perform operation).
  - (c) Push the result of expression into the stack top. e
4. If no element then  
pop the result  
else  
go to step 1.
5. Stop.

for example:-

$$P = 4, 5, 2, +, *, 16, 4, /, -$$

We can use above algorithm to evaluate it.

P = 4, 5, 2, +, \*, 16, 4, /, -

| Symbol    | Scan | Stack     |
|-----------|------|-----------|
|           | 4    | 4         |
|           | 5    | 4, 5      |
|           | 2    | 4, 5, 2   |
|           | +    | 4, 7      |
|           | *    | 28        |
|           | 16   | 28, 16    |
|           | 4    | 28, 16, 4 |
|           | /    | 28, 4     |
|           | -    | 24        |
| Result :- |      | 24        |

③ Conversion from INFIX to POSTFIX Notation:-

To convert infix expression to postfix expression we will use the stack data structure.

Input and Output

Input:-

The infix expression  $x^y / (5 * z) + 2$

Output:-

Postfix form is :-  $xy^5z*/2+$

/ /

## Algorithm:-

Step 1:- Push "(" on stack and ")" to end of infix expression.

Step 2:- Scan infix expression from left to right and repeat following steps until stack is empty.

Step 3:- If an operand is found add to P. (Postfix).

Step 4:- If an operator is found, then:-  
(a) repeatedly pop all the operators from stack which are having higher or same precedence and put them into the P.

(b) Push the new operator to stack.

Step 5:- If left parenthesis is found push on stack.

Step 6:- If right parenthesis is found, then:-  
(a) repeatedly pop the stack and add the popped operators to the expression P until a left parenthesis is found.  
(b) Remove the left parenthesis and don't add it in P.

Step 7:- Stop

Example: Infix Notation  $(P/(Q-R)) * S + T$

| Symbol | Scan | Stack | Postfix Expression |
|--------|------|-------|--------------------|
| (      |      | (     |                    |
| P      |      | (     | P                  |
| /      |      | (/    | P/                 |
| (      |      | ((    | P/(                |
| Q      |      | ((    | P/(Q               |
| -      |      | ((-   | P/(Q-              |
| R      |      | ((-   | P/(Q-R             |
| )      |      | (/    | P/(Q-R)            |
| *      |      | (*    | P/(Q-R)/           |
| S      |      | (*    | P/(Q-R)/S          |
| +      |      | (+    | P/(Q-R)/S*         |
| T      |      | (+    | P/(Q-R)/S*T        |
| )      |      |       | P/(Q-R)/S*T+       |

Postfix Expression:  $PQR- / S * T +$

#### ④ Parenthesis Matching:-

- A bracket is considered to be any one of the following characters: (, ), {, }, [, or ].
- Two brackets are considered to be a matched pair if the an opening bracket (i.e. [, {, or ( occurs to the left of a closing bracket (i.e. ), }, or ).

- Matched pairs of brackets :  $[\ ]$ ,  $\{ \}$ , and  $( )$ .
- $\{ [ ( ) ] \}$  is not balanced brackets, we can use a stack to store the unmatched brackets.

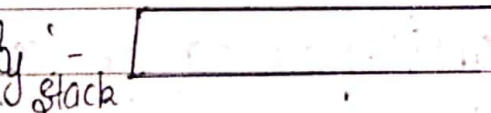
Algorithm:-

Scan the string from left to right,  
and for each char :-

1. If a left bracket, push onto stack.
2. If a right bracket, pop bracket from stack.  
(if not match or stack empty then fail).

Example:-

Initially -



str 

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| [ | { | ( | ) | } | ] |
|---|---|---|---|---|---|

 opening bracket.  
Push into stack.

Step 1:-

Stack 

|  |  |  |  |  |  |
|--|--|--|--|--|--|
|  |  |  |  |  |  |
|--|--|--|--|--|--|

str 

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| [ | { | ( | ) | } | ] |
|---|---|---|---|---|---|

  
↑  
Opening bracket push into stack.

Step 2:-

Stack [ | ε | ]

Str [ | { | ( | ) | } | ]

Opening bracket,  
push into stack.

Step 3:-

Stack [ | ε | ( | ]

Str [ | { | ( | ) | } | ]

Closing bracket.  
Check top of the  
stack is same  
kind or not.

Step 4:-

Stack [ | ε | ]

Str [ | { | ( | ) | } | ]

Closing bracket  
check top of the  
stack is same kind  
or not.

Step 5:-

Stack [ | ε | ]

Str [ | { | ( | ) | } | ]

Closing bracket.  
check top of stack  
is same kind  
or not.

## ⑤ Recursion:-

- Recursion is a process of calling the function by itself and function is called recursive function.
- We can easily describe the implementation of recursion an application of stack using some of examples that use the concept of recursion.

\_/\_/\_

Example of Recursion:-

1. Factorial of positive integer.
2. Towers of Hanoi

1. Factorial of a Positive integer:-

factorial is a mathematical term.

factorial of a number, say  $n$ , is equal to the product of all integers from 1 to  $n$ .

factorial of  $n$  is denoted by:-

$$n! = 1 \times 2 \times 3 \times \dots \times n.$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n=0 \text{ or } n=1 \\ n * (n-1)! & \text{if } n > 1 \end{cases}$$

Algorithm:- factorial (N).

Step 1:- if  $N=0$  then  
Return (1)

else

Return ( $N * \text{factorial}(N-1)$ )

[end if]

Step 2:- End.

## 2. Towers of Hanoi :-

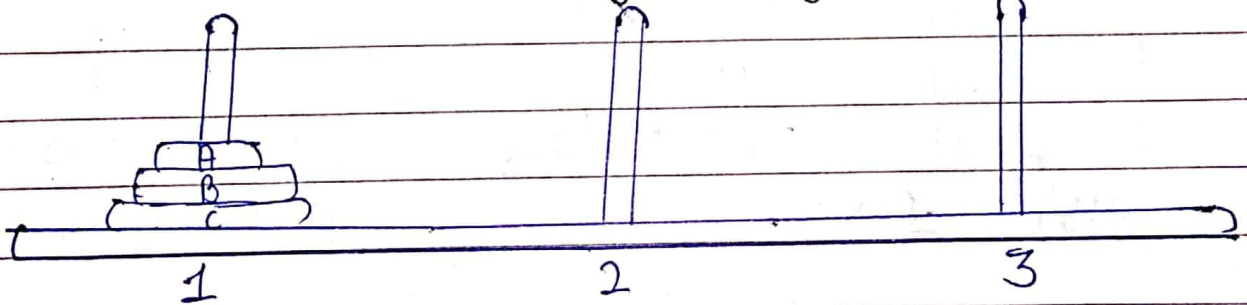
No discussion of recursion would be complete without the towers of Hanoi problem.

We are given a tower of  $N$  disks, initially stacked in increasing size on one of three poles.

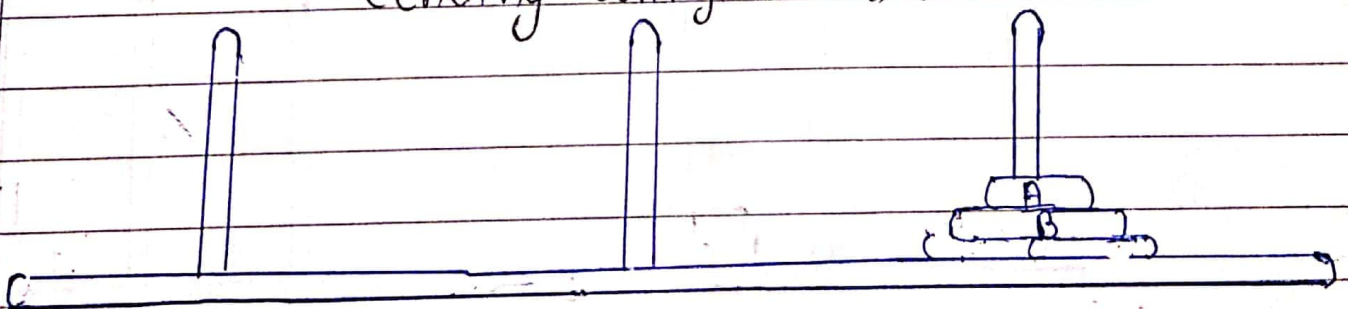
The objective is to transfer the entire tower to one of the other poles.

Rules:- (a) Move only one disc at a time.  
(b) Only the top disc can be moved.  
(c) A larger disc can not be placed on a smaller disc.

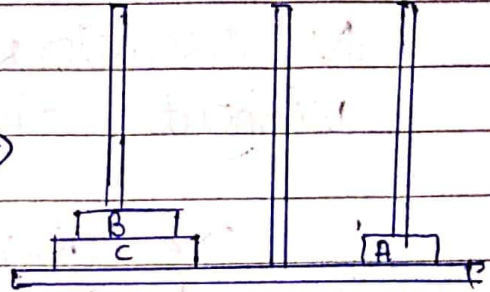
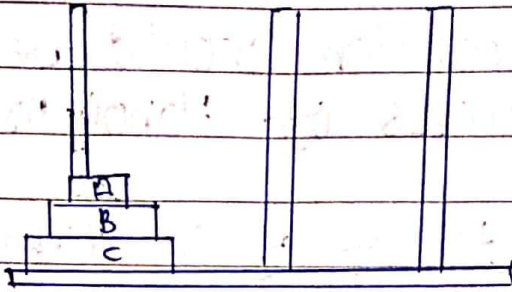
(Starting Configuration)



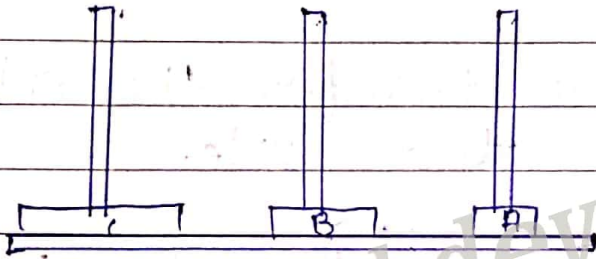
(Ending Configuration)



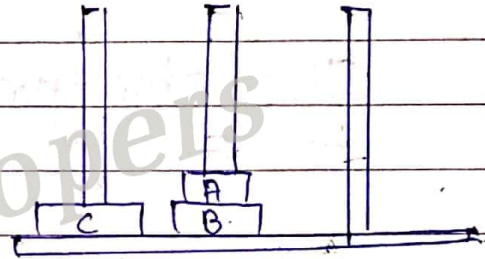
Solution:



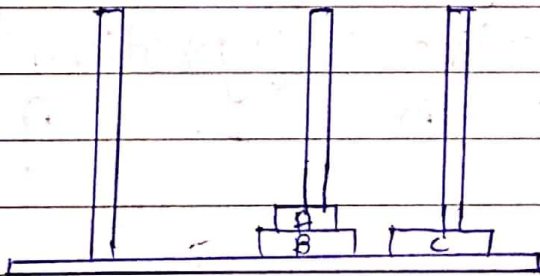
(i)  $1 \rightarrow 3$



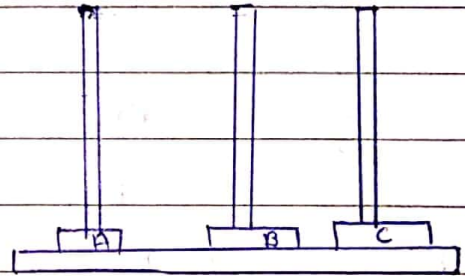
(ii)  $1 \rightarrow 2$



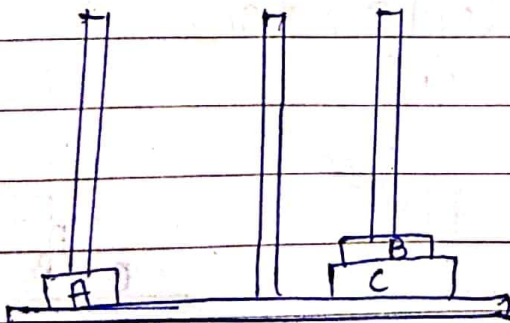
(iii)  $3 \rightarrow 2$



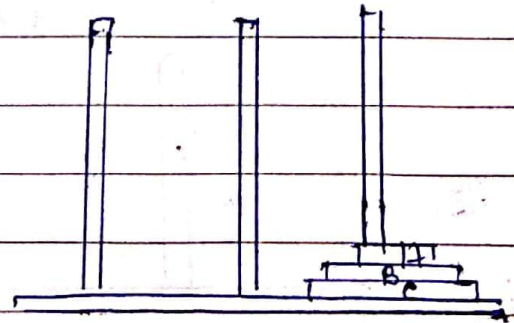
(iv)  $1 \rightarrow 3$



(v)  $2 \rightarrow 1$



(vi)  $2 \rightarrow 3$



(vii)  $1 \rightarrow 3$

