Data Structure (CS-303)

STACK

STACK

Stack is an important data structure which stores its elements in an ordered manner.

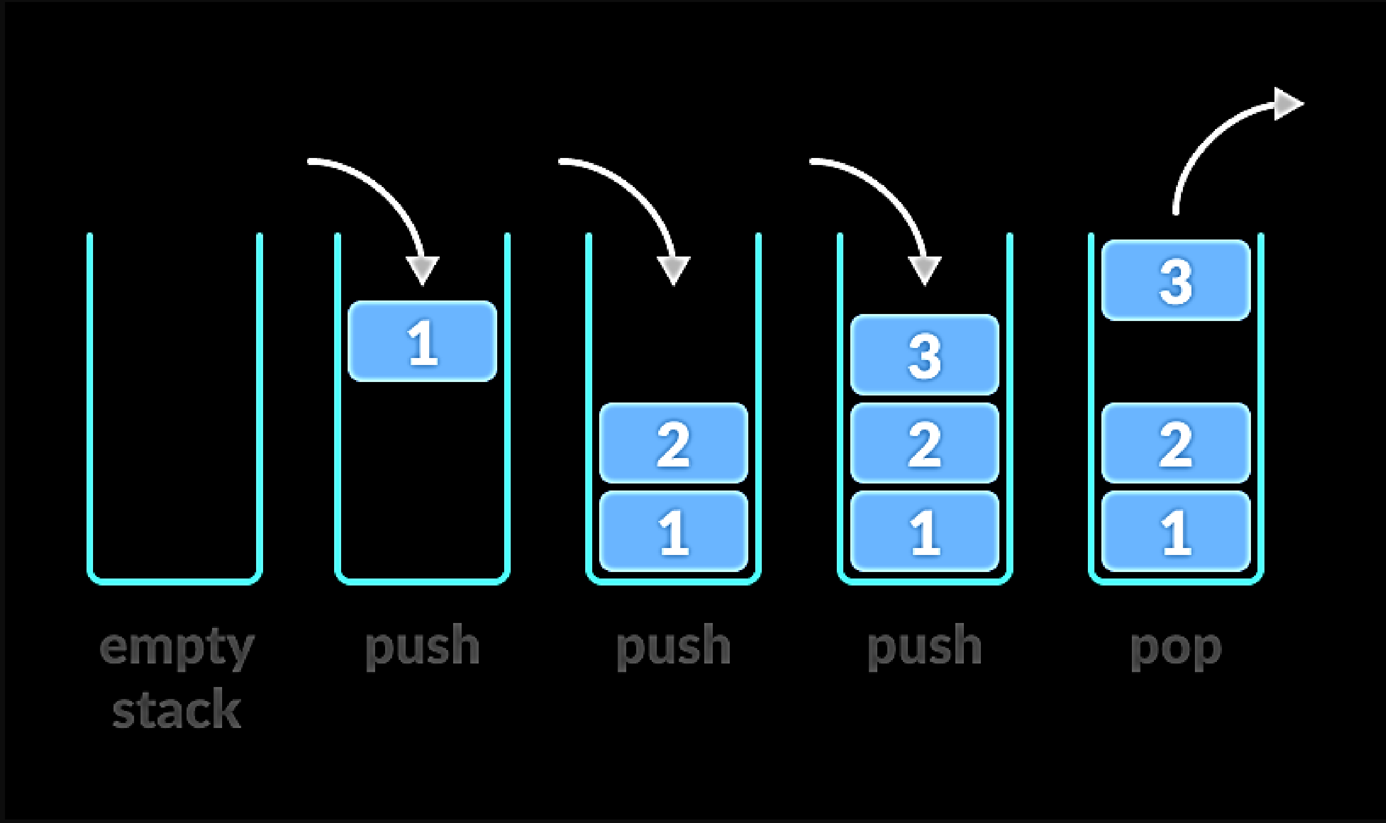
A stack is a linear structure in which items may be added or removed on lay at one end.

STACK

OPERATIONS ON A STACK

A stack supports three basic operations: push, pop, and peek.

- The push operation adds an element to the top of the stack.
- The pop operation removes the element from the top of the stack.
- The peek operation returns the value of the topmost element of the stack.



STACK

- Push is the term used to insert an element into a stack.
- Pop is the term used to delete an element from a stack

STACK: REPRESENTATION

Stack may be represented in following ways:

- A linear array
- A one-way list

STACK: ARRAY REPRESENTATION

Requirements:

- A linear array STACK
  - To maintain the stack
- A pointer variable TOP
  - To contain the location of the top element of the stack
- A variable MAXSTK
  - To store the maximum number of elements that can be held by the stack

STACK: ARRAY REPRESENTATION

Concepts

- If TOP=0 or TOP=NULL, stack is empty (UNDERFLOW condition)
- If TOP=MAXSTK, stack is already filled.

STACK: ARRAY REPRESENTATION

PUSH (STACK, TOP, MAXSTK, VAL)

```
Step 1:    If TOP = MAXSTK, then
               Write OVERFLOW,
               Goto Step 4
           [End of If]
Step 2:    Set TOP = TOP + 1
Step 3:    Set STACK [TOP] = VAL
Step 4:    Exit
```

STACK: ARRAY REPRESENTATION

POP (STACK, TOP, VAL)

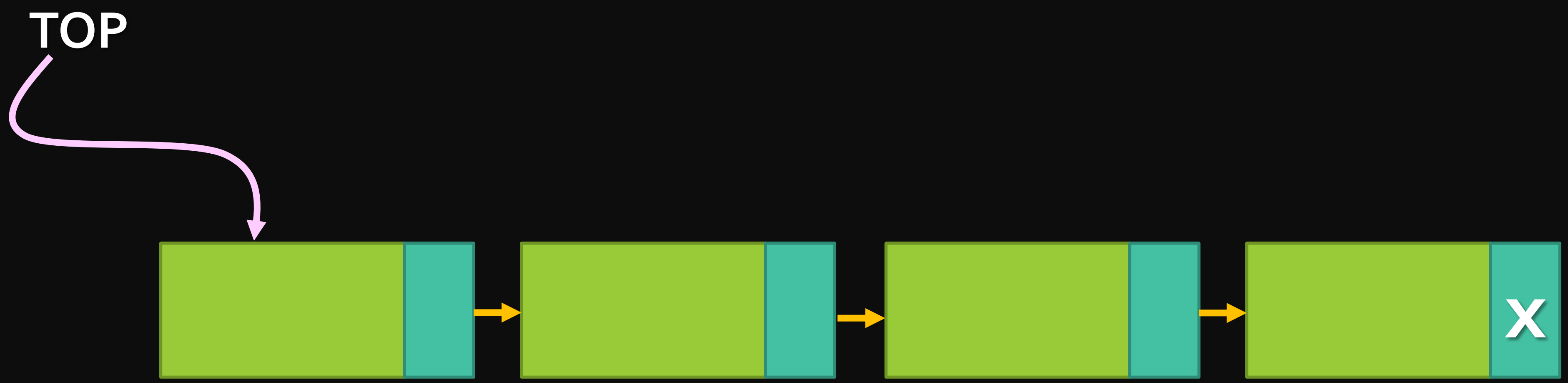
```
Step 1:     If TOP = NULL, then
                Write UNDERFLOW,
                Goto Step 4
        [End of If]
Step 2:     Set VAL = STACK [TOP]
Step 3:     Set TOP = TOP - 1
Step 4:     Exit
```

STACK: ARRAY REPRESENTATION

PEEK (STACK, TOP, VAL)

```
Step 1:        If TOP = NULL, then
                   Write UNDERFLOW,
                   Goto Step 3
               [End of If]
Step 2:        Write STACK [TOP]
Step 3:        Exit
```

STACK: LINKED REPRESENTATION



STACK: LINKED REPRESENTATION

```
PUSH (INFO, NEXT, TOP, AVAIL, VAL)
Step 1:     If AVAIL = NULL, the
                 Write OVERFLOW, and GoTo Step 7
Step 2:     Set NEW_NODE = AVAIL
Step 3:     Set AVAIL = NEXT [AVAIL]
Step 4:     Set INFO[NEW_NODE] = VAL
Step 5:     Set NEXT[NEW_NODE] = TOP
Step 6:     Set TOP = NEW
Step 7:     Exit
```

STACK: LINKED REPRESENTATION

```
POP (INFO, NEXT, TOP, AVAIL, VAL)
Step 1:     If TOP = NULL, then
                Write UNDERFLOW, and GoTo Step 7
Step 2:     Set VAL = INFO[TOP]
Step 3:     Set TEMP = TOP
Step 4:     Set TOP = NEXT[TOP]
Step 5:     Set NEXT[TEMP] = AVAIL
Step 6:     Set AVAIL = TEMP
Step 7:     Exit
```

STACK: RECURSION

A recursive function is defined as a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself. Since a recursive function repeatedly calls itself, it makes use of the system stack to temporarily store the return address and local variables of the calling function. Every recursive solution has two major cases. They are

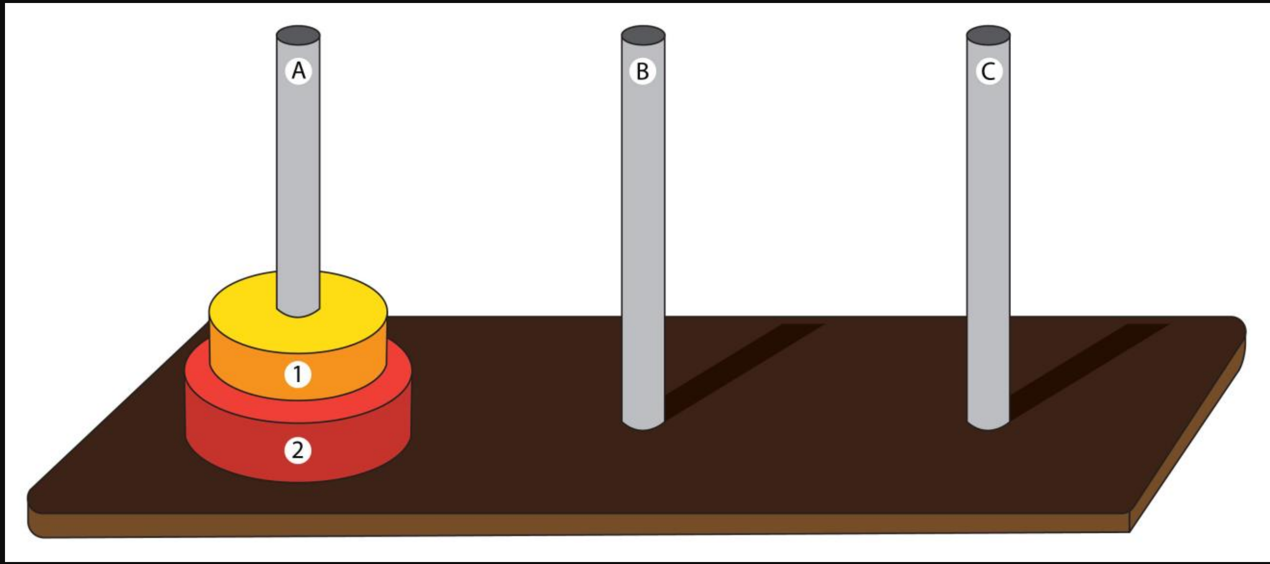
- Base case, in which the problem is simple enough to be solved directly without making any further calls to the same function.
- Recursive case, in which first the problem at hand is divided into simpler sub-parts. Second the function calls itself but with sub-parts of the problem obtained in the first step. Third, the result is obtained by combining the solutions of simpler sub-parts.

STACK: RECURSION (TOWER OF HANOI)

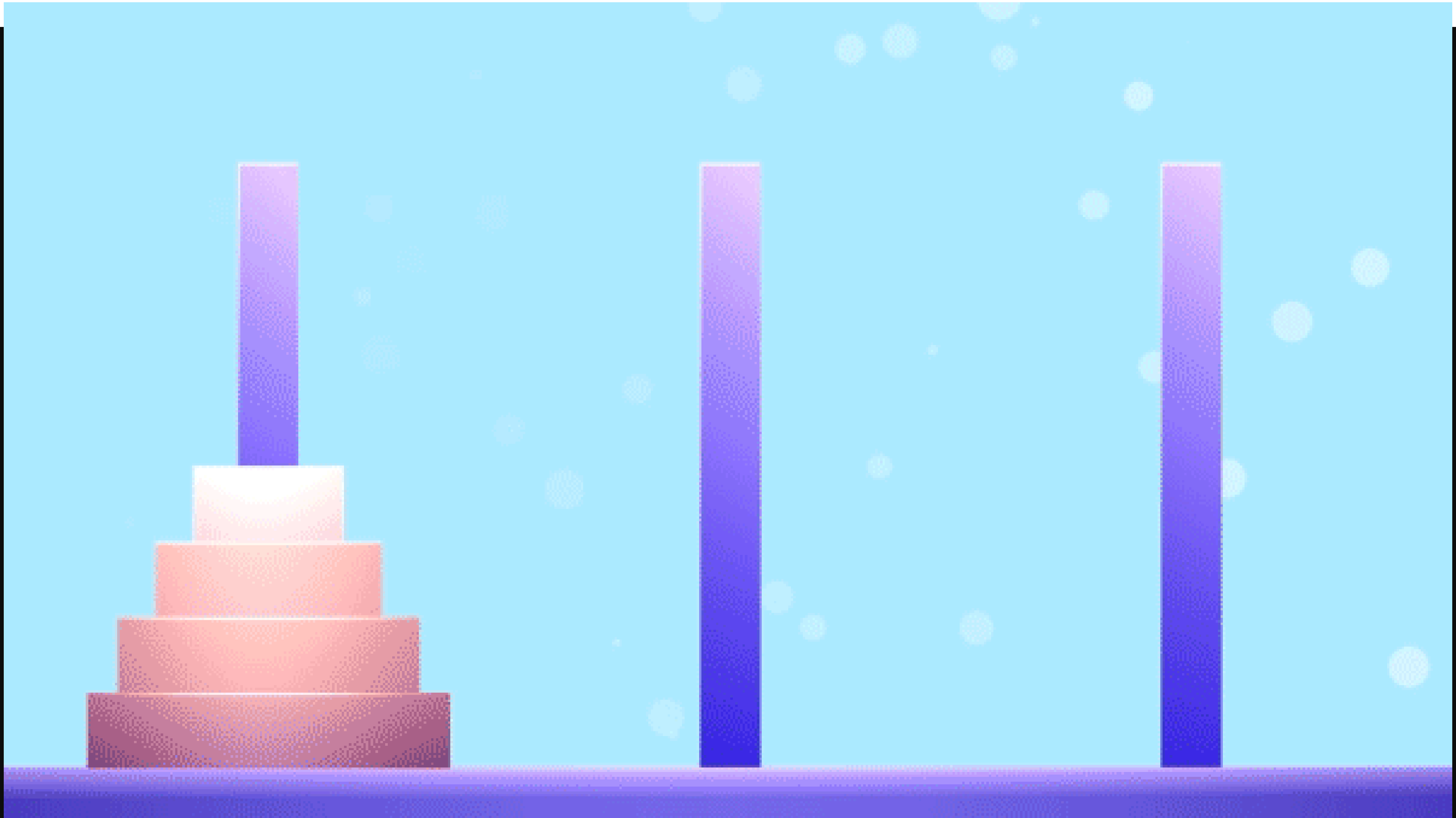
Tower of Hanoi consists of three pegs or towers with n disks placed one over the other.

The objective of the puzzle is to move the stack to another peg following these simple rules.

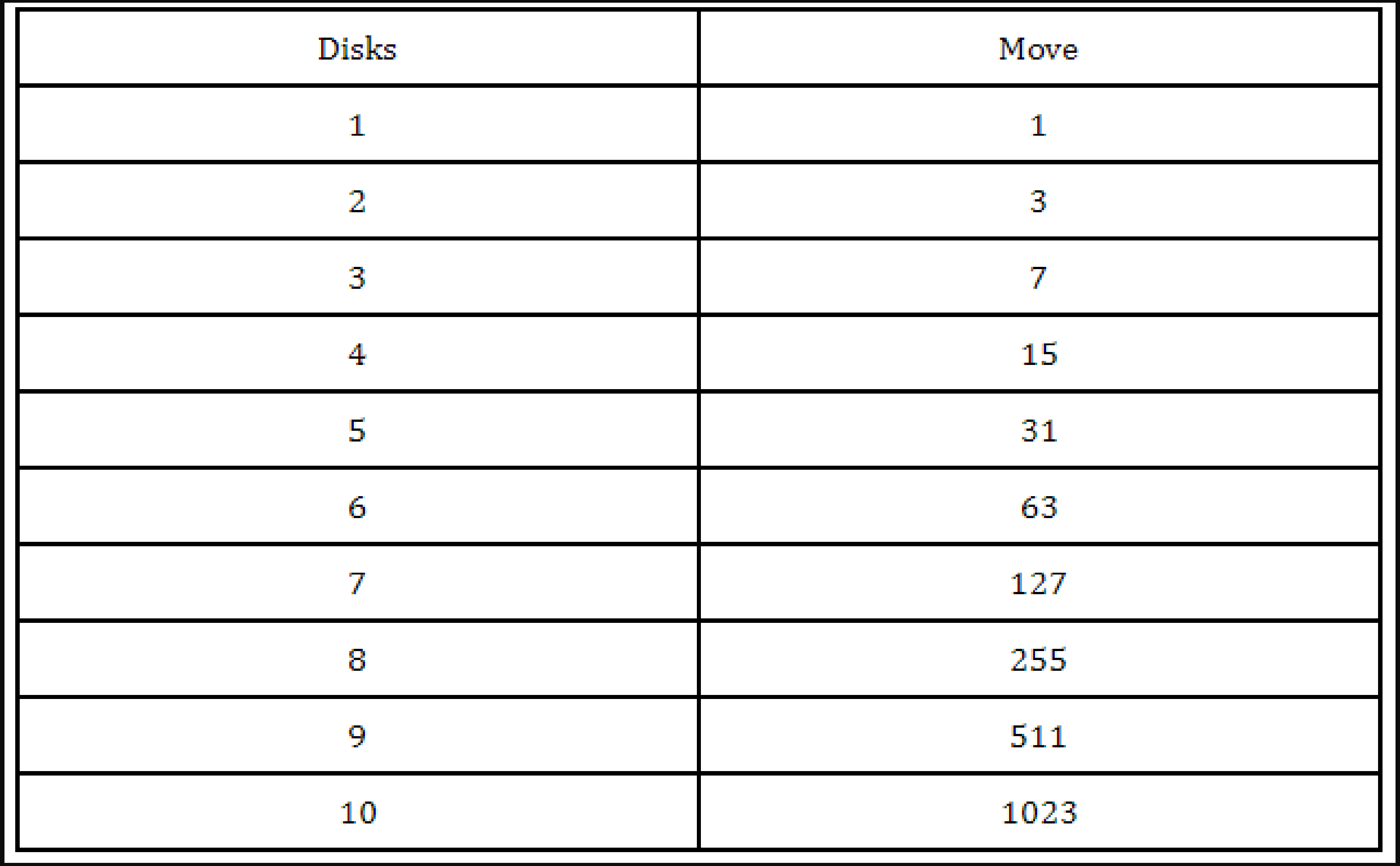
- Only one disk can be moved at a time.
- No disk can be placed on top of the smaller disk.



STACK: RECURSION (TOWER OF HANOI)



STACK: RECURSION (TOWER OF HANOI)



Disks	Move
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

STACK: RECURSION (TOWER OF HANOI)

Suppose that there are 64 disks and one can move these disks at a speed of 1 per second.

At this speed, they would need 2^{64} -1 move to complete the task.

That is, 18,446,744,073,709,551,615 moves to complete, which would take about 21,35,03,98,23,34,601 days or 5,84,94,24,17,355 years (580 billion years).

Considering our Milky Way is about 13.21 billion years old and our planet is only 5 billion years old, that is a lot of time for the prophecy to be true.

STACK: POLISH NOTATIONS

Infix, postfix, and prefix notations are three different but equivalent notations of writing algebraic expressions.

STACK: POLISH NOTATIONS

Infix Notations

While writing an arithmetic expression using infix notation, the operator is placed in between the operands. For example, A+B; here, plus operator is placed between the two operands A and B. Although it is easy for us to write expressions using infix notation, computers find it difficult to parse as the computer needs a lot of information to evaluate the expression. Information is needed about operator precedence and associativity rules, and brackets which override these rules. So, computers work more efficiently with expressions written using prefix and postfix notations.

STACK: POLISH NOTATIONS

Postfix Notations

Postfix notation was developed by Jan Łukasiewicz who was a Polish logician, mathematician, and philosopher. His aim was to develop a parenthesis-free prefix notation (also known as Polish notation) and a postfix notation, which is better known as Reverse Polish Notation or RPN.

In postfix notation, as the name suggests, the operator is placed after the operands. For example, if an expression is written as A+B in infix notation, the same expression can be written as AB+ in postfix notation. The order of evaluation of a postfix expression is always from left to right. Even brackets cannot alter the order of evaluation.

The expression (A + B) * C can be written as:

[AB+]*C

AB+C* in the postfix notation

STACK: POLISH NOTATIONS

Postfix Notations

A postfix operation does not even follow the rules of operator precedence. The operator which occurs first in the expression is operated first on the operands. For example, given a postfix notation AB+C*. While evaluation, addition will be performed prior to multiplication. Thus we see that in a postfix notation, operators are applied to the operands that are immediately left to them. In the example, AB+C*, + is applied on A and B, then * is applied on the result of addition and C.

Although a prefix notation is also evaluated from left to right, the only difference between a postfix notation and a prefix notation is that in a prefix notation, the operator is placed before the operands. For example, if A+B is an expression in infix notation, then the corresponding expression in prefix notation is given by +AB.

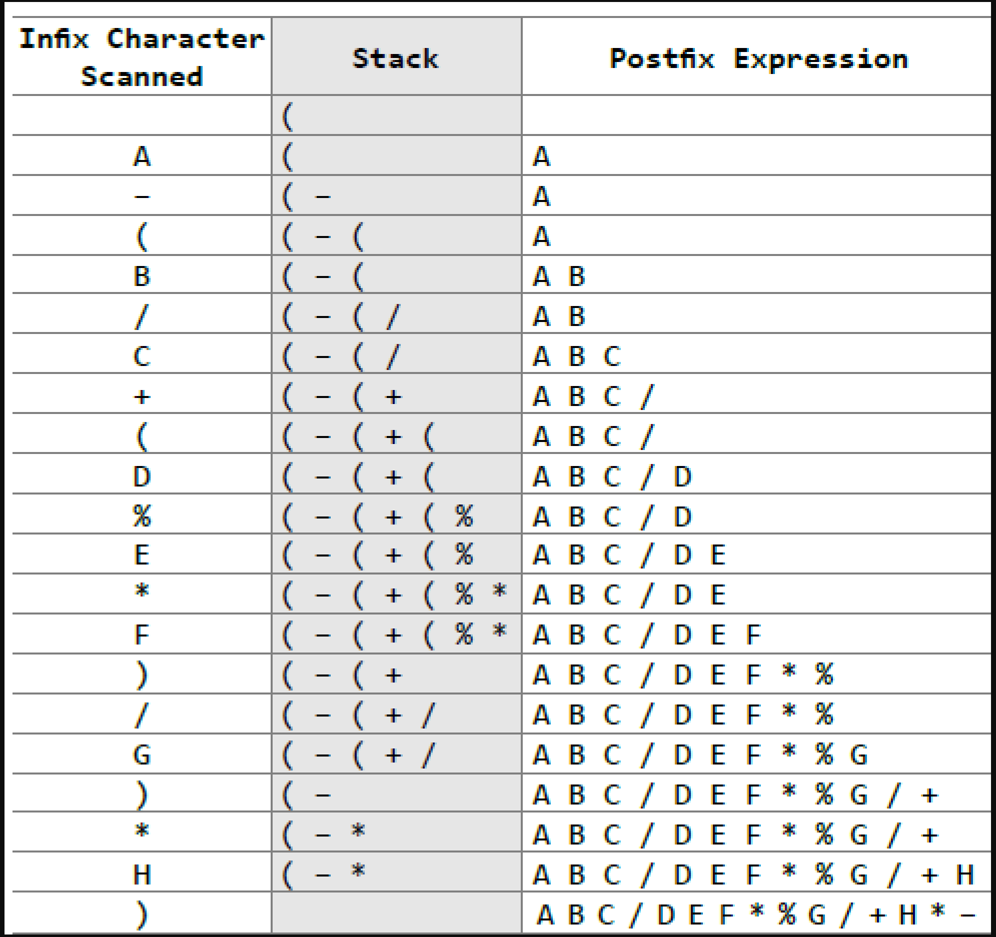
While evaluating a prefix expression, the operators are applied to the operands that are present immediately on the right of the operator. Like postfix, prefix expressions also do not follow the rules of operator precedence and associativity, and even brackets cannot alter the order of evaluation.

STACK: POLISH NOTATIONS

(2+4) * (4+6)

2	4	+	4	6	+	*
				6		
	4		4	4	10	
2	2	6	6	6	6	60

STACK: POLISH NOTATIONS



Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
–	(–	A
(	(– (	A
B	(– (	A B
/	(– (/	A B
C	(– (/	A B C
+	(– (+	A B C /
(	(– (+ (	A B C /
D	(– (+ (	A B C / D
%	(– (+ (%	A B C / D
E	(– (+ (%	A B C / D E
*	(– (+ (% *	A B C / D E
F	(– (+ (% *	A B C / D E F
)	(– (+	A B C / D E F * %
/	(– (+ /	A B C / D E F * %
G	(– (+ /	A B C / D E F * % G
)	(–	A B C / D E F * % G / +
*	(– *	A B C / D E F * % G / +
H	(– *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * –

- A – (B / C + (D % E * F) / G)* H
- (A – (B / C + (D % E * F) / G)* H)

STACK: POLISH NOTATIONS

S.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

STACK: POLISH NOTATIONS

The procedure for getting the result is:

i. Convert the expression in Reverse Polish notation(post-fix notation).

ii. Push the operands into the stack in the order they are appear.

iii. When any operator encounter then pop two topmost operands for executing the operation.

iv. After execution push the result obtained into the stack.

v. After the complete execution of expression the final result remains on the top of the stack.

STACK: POLISH NOTATIONS POSTFIX EVALUATION ALGORITHM

Step 1 – scan the expression from left to right

Step 2 – if it is an operand push it to stack

Step 3 – if it is an operator pull operand from stack and perform operation

Step 4 – store the output of step 3, back to stack

Step 5 – scan the expression until all operands are consumed

Step 6 – pop the stack and perform operation

STACK: POLISH NOTATIONS

ALGORITHM TO CONVERT AN INFIX NOTATION TO POSTFIX NOTATION

```
Step 1:   Add ) to the end of the infix expression
Step 2:   Push ( on to the stack
Step 3:   Repeat until each character in the infix notation is scanned
              IF a ( is encountered, push it on the stack
              IF an operand (whether a digit or a character) is encountered, add it postfix expression.
              IF a ) is encountered, then
                  a. Repeatedly pop from stack and add it to the postfix expression until a ( is encountered.
                  b. Discard the ( . That is, remove the ( from stack and do not add it to the postfix expression
              IF an operator is encountered, then
                  a. Repeatedly pop from stack and add each operator (popped from the stack) to the
                     postfix expression which has the same precedence or a higher precedence than
                  b. Push the operator to the stack
              [END OF IF]
Step 4:   Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5:   EXIT
```

STACK: POLISH NOTATIONS DO IT YOURSELF

A^B*C-D+E/F/(G+H)

STACK: POLISH NOTATIONS DO IT YOURSELF

(A+B)*(C^)D-E)+F)-G