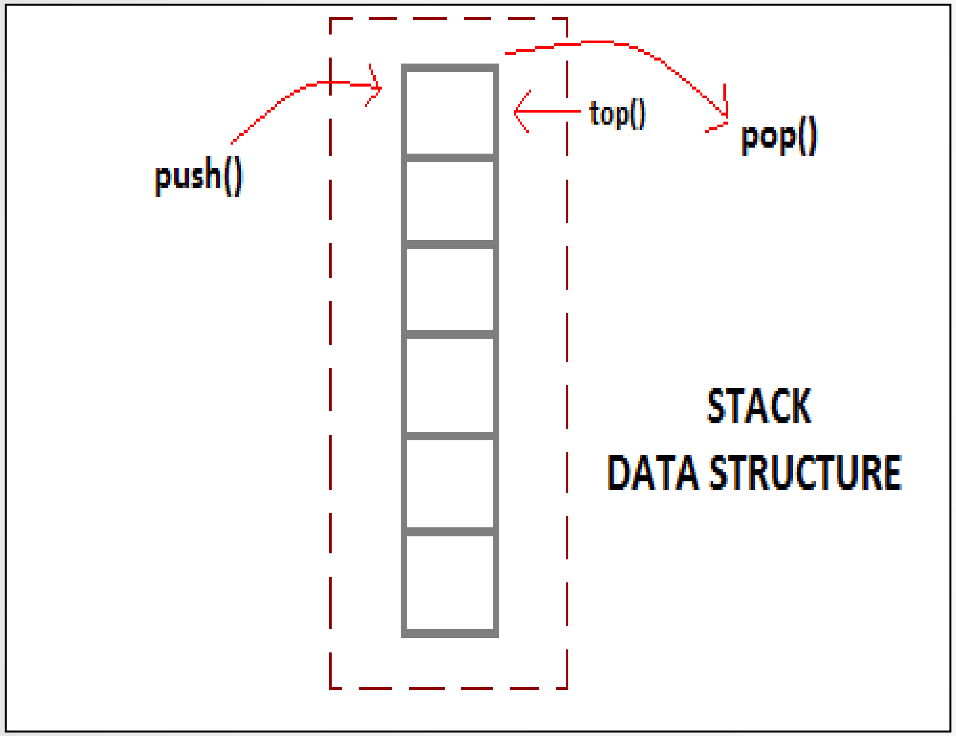
Stack:

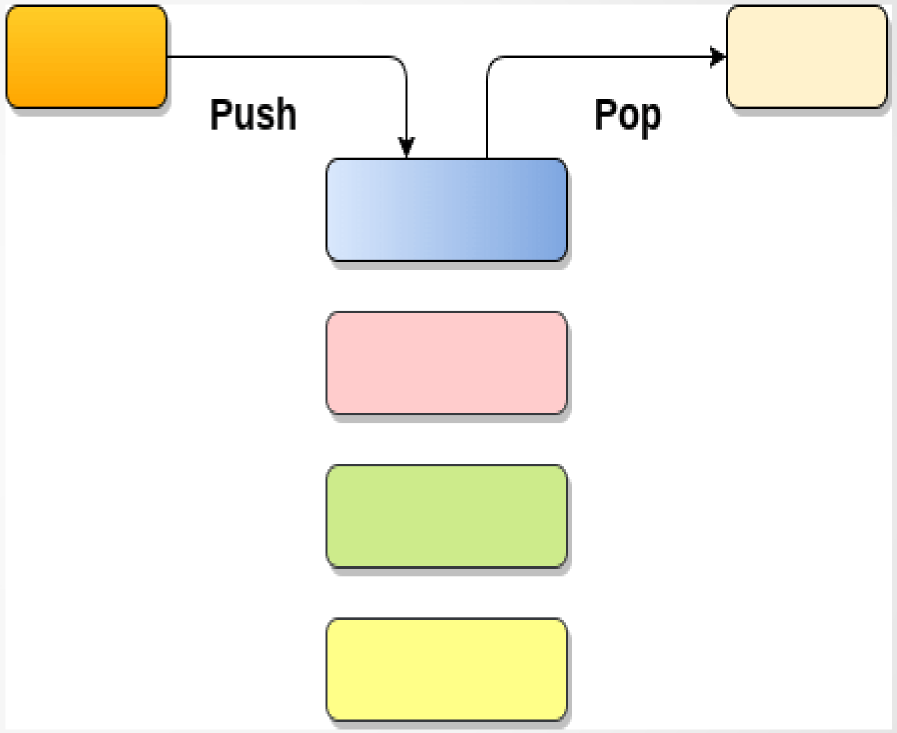
- Introduction To Stacks
- Representation of Stacks through Arrays
- Representation of Stacks through Linked List
- Applications of Stacks
- Conversion of Infix to Postfix Expression
- Evaluation of Postfix Expression
- Conversion of Infix to Prefix Expression
- Evaluation of Prefix Expression

Introduction to Stack

- Stack is an abstract data type with a predefined capacity.
- A stack is a linear data structure.
- A stack is an ordered collection of data elements where the insertion and deletion operations take place at one end only
- Its called Last In First Out()
- It is a simple data structure that allows adding and removing elements in a particular order.
- Every time an element is added, it goes on the top of the stack, the only element that can be removed is the element that was at the top of the stack, just like a pile of objects

Introduction to Stack



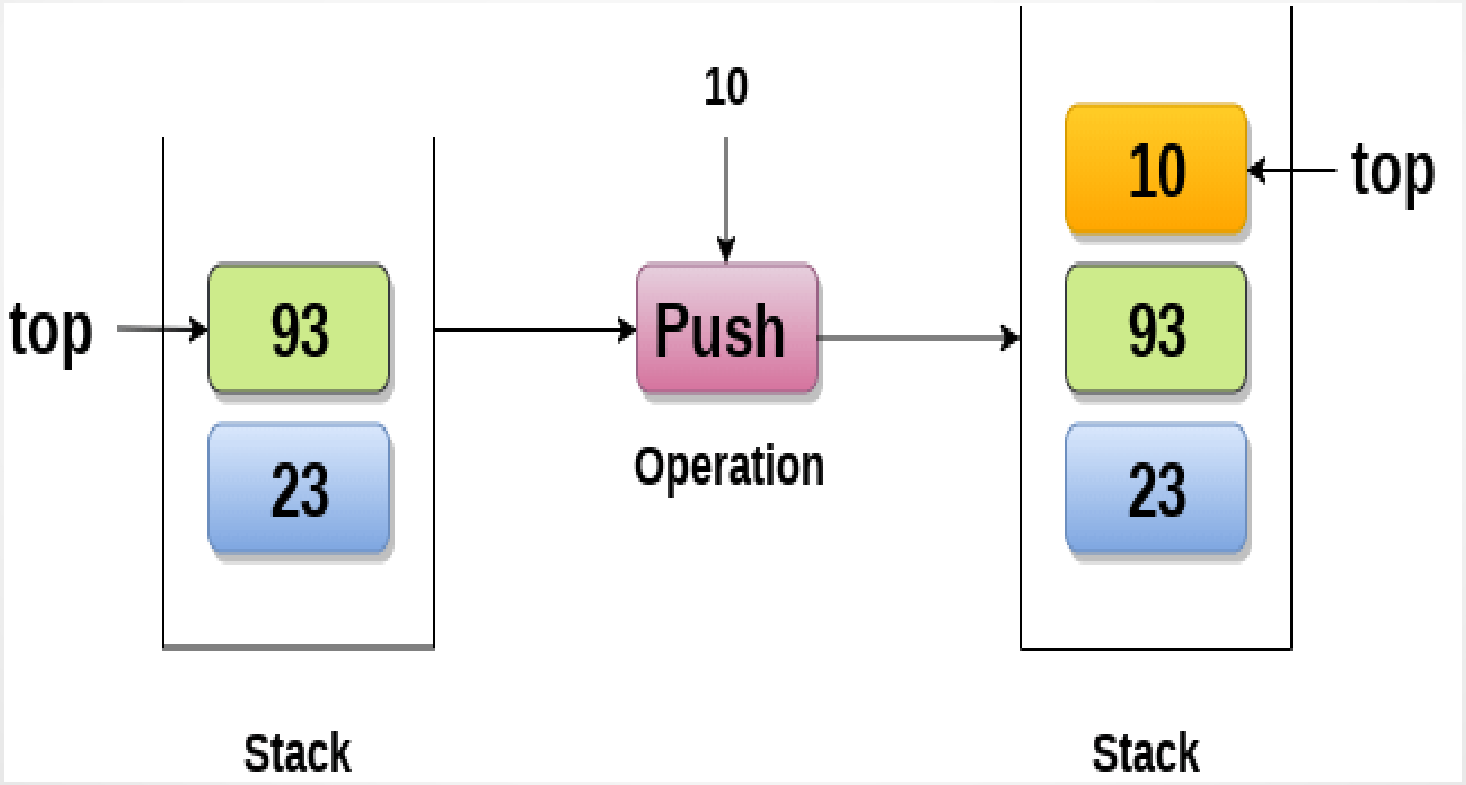


Basic features of Stack

- Stack is **an ordered list of similar data type** in which, **insertion and deletion can be performed only at one end that is called top.** .
- Stack is **a LIFO structure. (Last in First out).**
- **push()** function is used to insert new elements into the Stack and **pop()** is used to delete an element from the stack. Both insertion and deletion are allowed at only one end of Stack called Top.
- Stack is said to be in **Overflow** state when it is completely full and is said to be in **Underflow** state if it is completely empty.

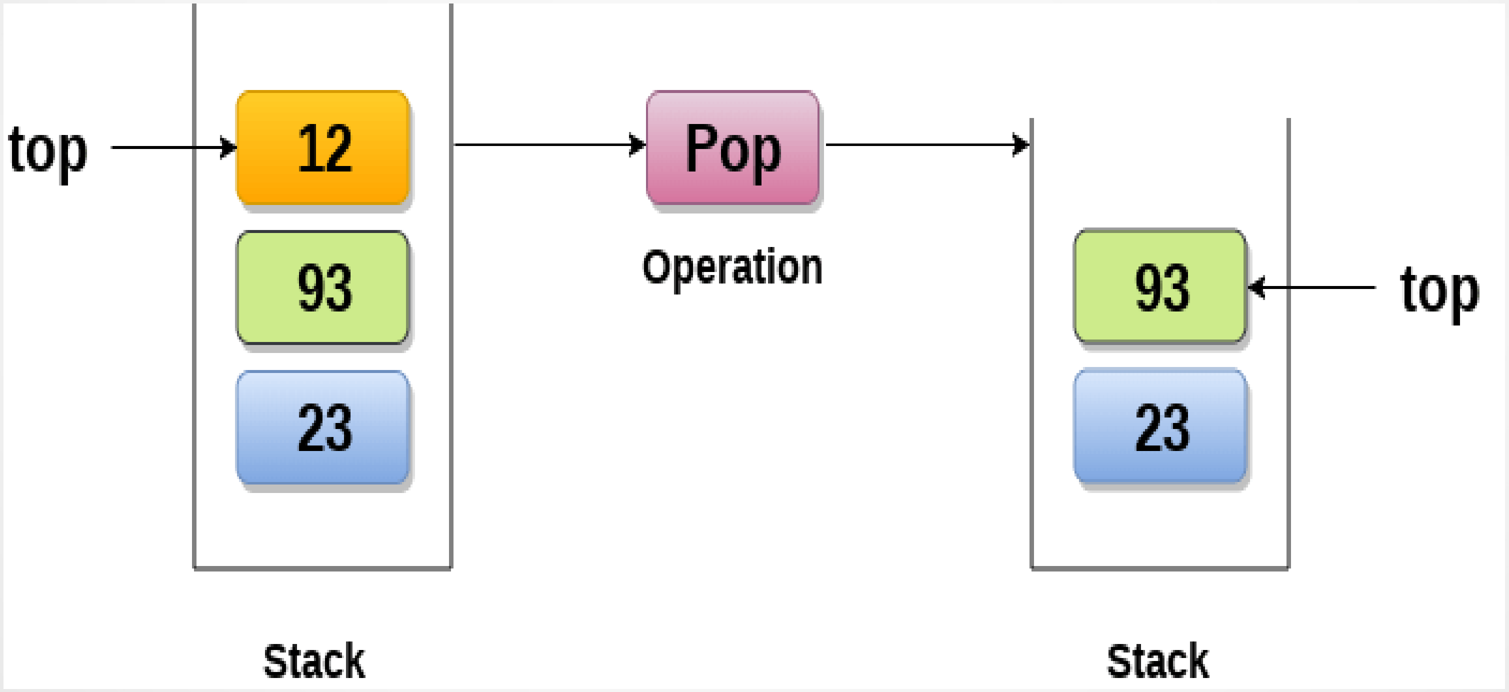
Operations on Stack

Push : Adding an element onto the stack



Operations on Stack

Pop: Removing an element from the stack



Operations on Stack

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks −

Peek()/peep() − get the top data element of the stack, without removing it.

isFull() − check if stack is full.

isEmpty() − check if stack is empty.

- At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. **The top pointer provides top value of the stack without actually removing it.**

Application of Stack

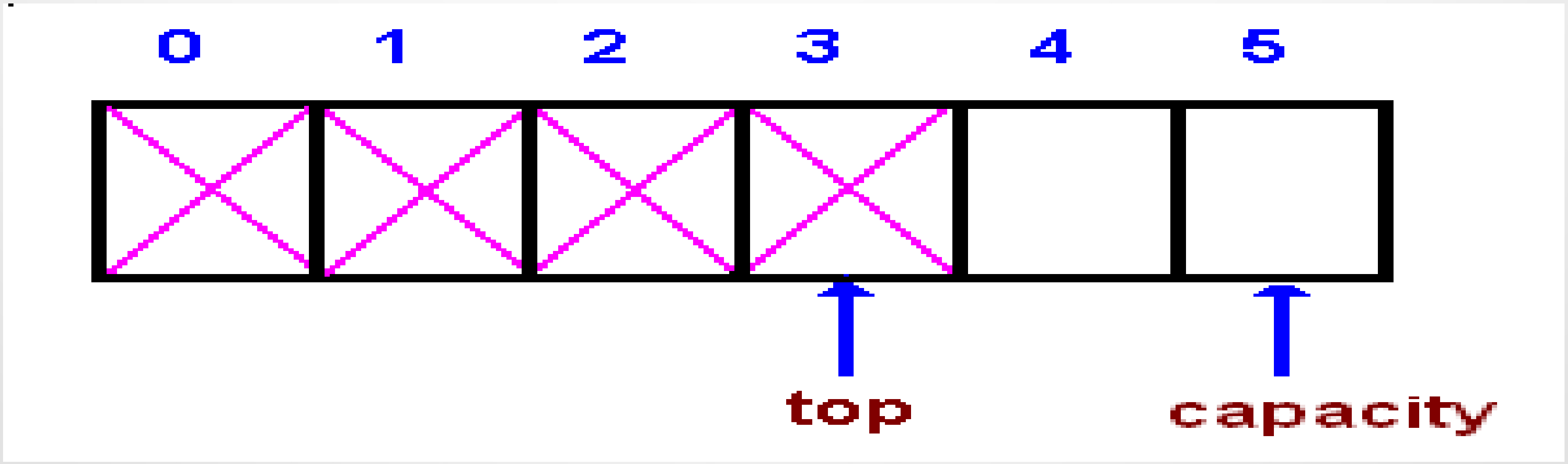
- Recursion
- Expression evaluations and conversions
- Parsing
- Browsers
- Tree Traversals

Implementation of Stack

- Stack can be easily implemented using an Array or a Linked List.
- Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and de allocate, but is not limited in size.

Representation of Stacks through Arrays

- In array implementation, the stack is formed by using the array. All the operations regarding the stack are performed using arrays.
- In an array-based implementation we maintain the following fields: an array A of a default size (≥ 1), the variable top that refers to the top element in the stack and the capacity that refers to the array size.



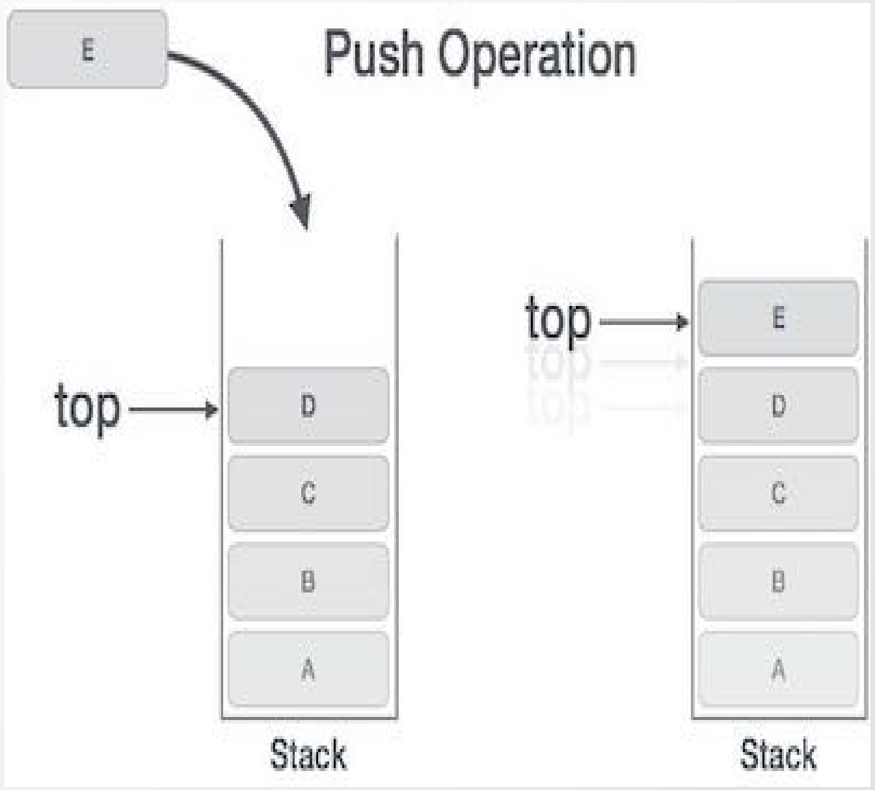
Representation of Stacks through Arrays

- Every stack has a variable called TOP associated with it.
- TOP is used to store the address of the top most element of the stack.
- It is in this position where the element will be added or deleted.
- There is another variable called MAX which is used to store the maximum numbers of elements that the stack can hold.
- If TOP = NULL, then it indicates that the stack is empty.
- If TOP = MAX-1 , then it indicates that the stack is full.

Push Operation

- The process of putting a new data element onto stack is known as PUSH Operation.
- **Algorithm:**

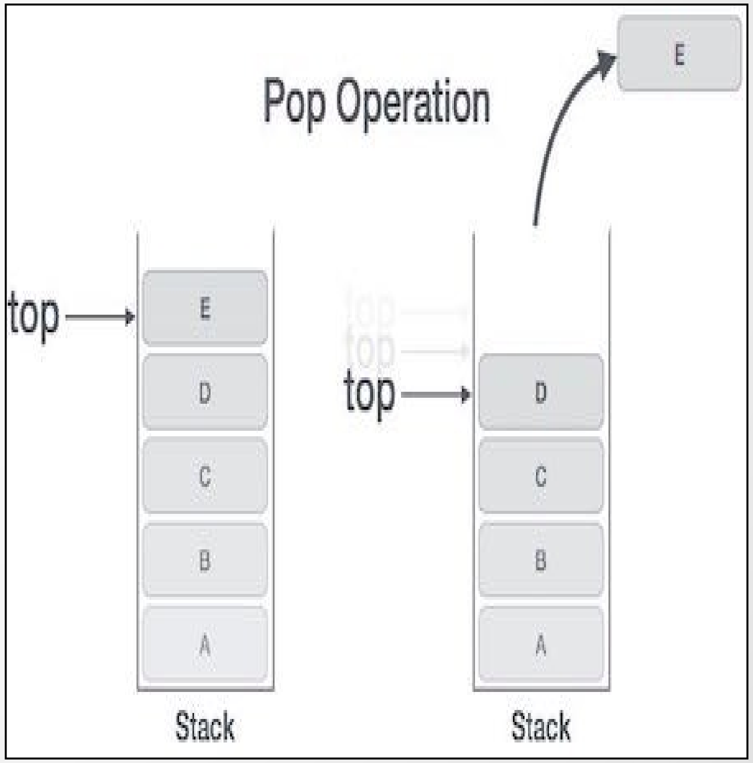
```
Step1: IF TOP = MAX-1
    then PRINT “OVERFLOW”
    [END OF IF]
Step2: SET TOP = TOP  + 1
Step3: SET STACK[TOP] = VALUE
Step4: END
```



Pop Operation

- The Pop operation is used to delete the top most element from stack.
- **Algorithm:**

```
Step1: IF TOP = NULL
    then PRINT "UNDERFLOW"
    [END OF IF]
Step2: SET VAL= STACK[TOP]
Step3: SET TOP = TOP - 1
Step4: END
```



Peep Operation

- The Peep operation returns the value of the topmost of the element of the stack without deleting it from the stack.
- **Algorithm:**

```
Step1: IF TOP = NULL
      then PRINT "STACK IS EMPTY"
      [END OF IF]
Step2: RETURN STACK[TOP]
Step3 END
```

Peep Operation

- The Peep operation returns the value of the topmost of the element of the stack without deleting it from the stack.
- **Algorithm:**

```
Step1: IF TOP = NULL
    then PRINT “STACK IS EMPTY”
    [END OF IF]
Step2: RETURN STACK[TOP]
Step3 END
```

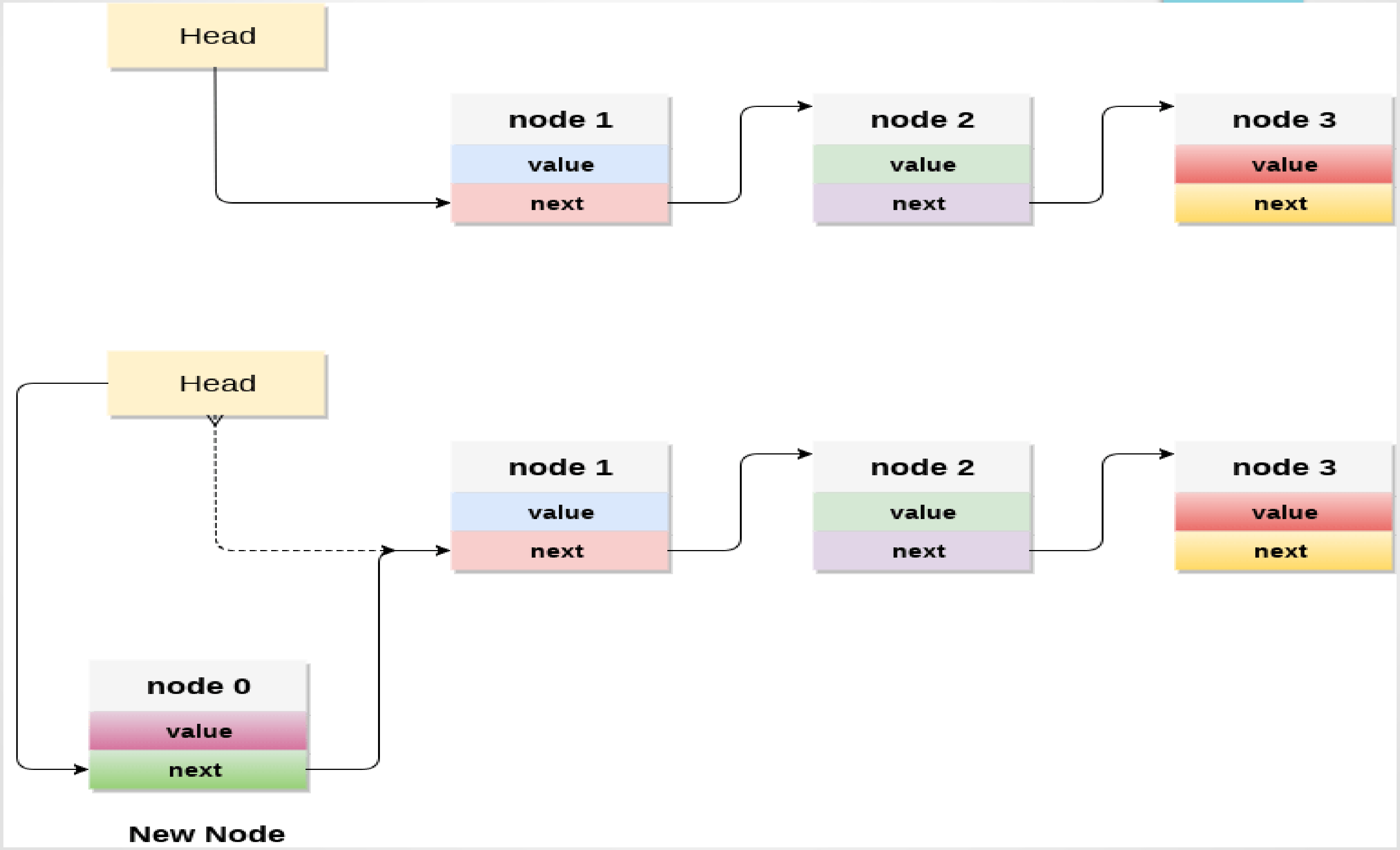
Representation of Stacks through Link List

- Instead of using array, we can also use linked list to implement stack. Linked list allocates the memory dynamically. However, time complexity in both the scenario is same for all the operations i.e. push, pop and peek.
- In linked list implementation of stack, the nodes are maintained non-contiguously in the memory.
- Each node contains a pointer to its immediate successor node in the stack.
- Stack is said to be overflown if the space left in the memory heap is not enough to create a node.

Representation of Stacks through Link List

- In a linked stack, every node has two parts-
- One that stores data and another that stores the address of the next node.
- The START pointer of the linked list is used as TOP.
- All insertion and deletions are done at the node pointed by the TOP.
- If TOP= NULL, then it indicates that the stack is empty.

Push Operation



Push Operation

```
Step1: SET New_Node=AVAIL
Step2: SET New_Node->DATA=VAL
Step3: IF TOP=NULL,then
            SET New_Node->NEXT=NULL
            SET TOP = New_Node
            ELSE
            SET New_Node->NEXT = TOP
            SET TOP = New_Node
        [END OF IF]
Step4: END
```

Pop Operation

```
Step1: IF TOP = NULL, then
        PRINT "UNDERFLOW"
    [END OF IF]
Step2: SET PTR= TOP
Step3: SET TOP=TOP->NEXT
Step4: FREE PTR
Step5: END
```

INFIX,POSTFIX & PREFIX NOTATION

The way to write arithmetic expression is known as a notation. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are −

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

INFIX,POSTFIX & PREFIX NOTATION

Infix Notation:

We write expression in infix notation, e.g. a - b + c, where operators are used in-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

INFIX,POSTFIX & PREFIX NOTATION

Prefix Notation:

In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +ab. This is equivalent to its infix notation a + b. Prefix notation is also known as Polish Notation.

INFIX,POSTFIX & PREFIX NOTATION

Postfix Notation:

This notation style is known as Reversed Polish Notation. In this notation style, the operator is postfixed to the operands i.e., the operator is written after the operands. For example, ab+. This is equivalent to its infix notation a + b.

Precedence & Associativity of operators

- **Precedence of operators** come into pictures when in an expression we need to decide which operator will be evaluated first. Operator with higher precedence will be evaluated first.

- **Associativity of opearators** come into picture **when precedence of operators are same and we need to decide which operator will be evaluated first.**

Precedence & Associativity of operators

Sr.No.	Operator	Precedence	Associativity
1	Exponentiation ^	Highest	Right Associative
2	Multiplication (∗) & Division (/)	Second Highest	Left Associative
3	Addition (+) & Subtraction (−)	Lowest	Left Associative

INFIX,POSTFIX & PREFIX NOTATION

Sr. No	Infix	Prefix	Postfix
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 -

Conversion of Infix to Postfix Expression

- Algorithm:
- Step1: Read the infix expression from left to right one character at a time.
- Step2: Repeat step 3
- Step3: Read Symbol
  - a. If symbol is operand put into postfix string.
  - b. If symbol is left parenthesis push into stack.
  - c. If right parenthesis, pop top of stack until left parenthesis occur and make postfix expression.

Conversion of Infix to Postfix Expression

```
d. If operator then
    i. If stack is empty push operator
    ii. If the top of the stack is opening parenthesis, push
operator on stack.
    iii. if operator has same or less precedence then operators
        available at top of stack, then pop all such operators
        and from postfix string.
    iv. Push scanned/incoming operator to the stack
[END OF IF]
```

Conversion of Infix to Postfix Expression

Step4: Repeatedly pop from the stack and add it to thr postfix `

expression until stack is empty

Step5: Exit

Evaluate the Postfix Expression

- **Algorithm:**
- Step1: Add a “)” at the end of the postfix expression
- Step2: Scan every character of the postfix expression and

  repeat step 3 and 4 until “)” is encountered.
- Step3: If an operand is encountered, push it on the stack

Evaluate the Postfix Expression

```
    If an Operator O is encountered then,
    a. pop the top two elements from the stack as A and B
    b. Evaluate B O A, where A was the topmost element and B
was the element below A.
    c. Push the result of evaluation on the stack
    [END OF IF]
Step4: SET RESULT equal to the topmost element of the stack.
Step5: EXIT
```

Conversion of Infix to Prefix Expression

Step1: Reverse the infix string. Note while reversing the string, you must interchange left and right parenthesis

Step2: Obtain the corresponding postfix expression of the infix expression obtained as a result of step 1.

Step3: Reverse the postfix expression to get the prefix expression

Conversion of Infix to Prefix Expression

Example:

Step 1: Reverse the infix expression i.e A+B*C will become C*B+A. Note while reversing each ‘(‘ will become ‘)’ and each ‘)’ becomes ‘(‘.

Step 2: Obtain the postfix expression of the modified expression i.e CB*A+.

Step 3: Reverse the postfix expression. Hence in our example prefix is +A*BC