Stack

(Stack, Arithmetic Expression)

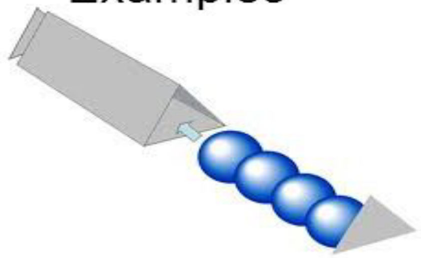
Stack

- It is a linear data structure consisting of list of items.
- In stack, data elements are added or removed only at one end, called ***the top of the stack***.

Stack

Examples

Stack of Books


Box of Tennis Balls
(Closed at 1 end)

Stack of Plates

Stack of Clothes.

Trains in a railway yard.

Goods in a cargo.

Type:

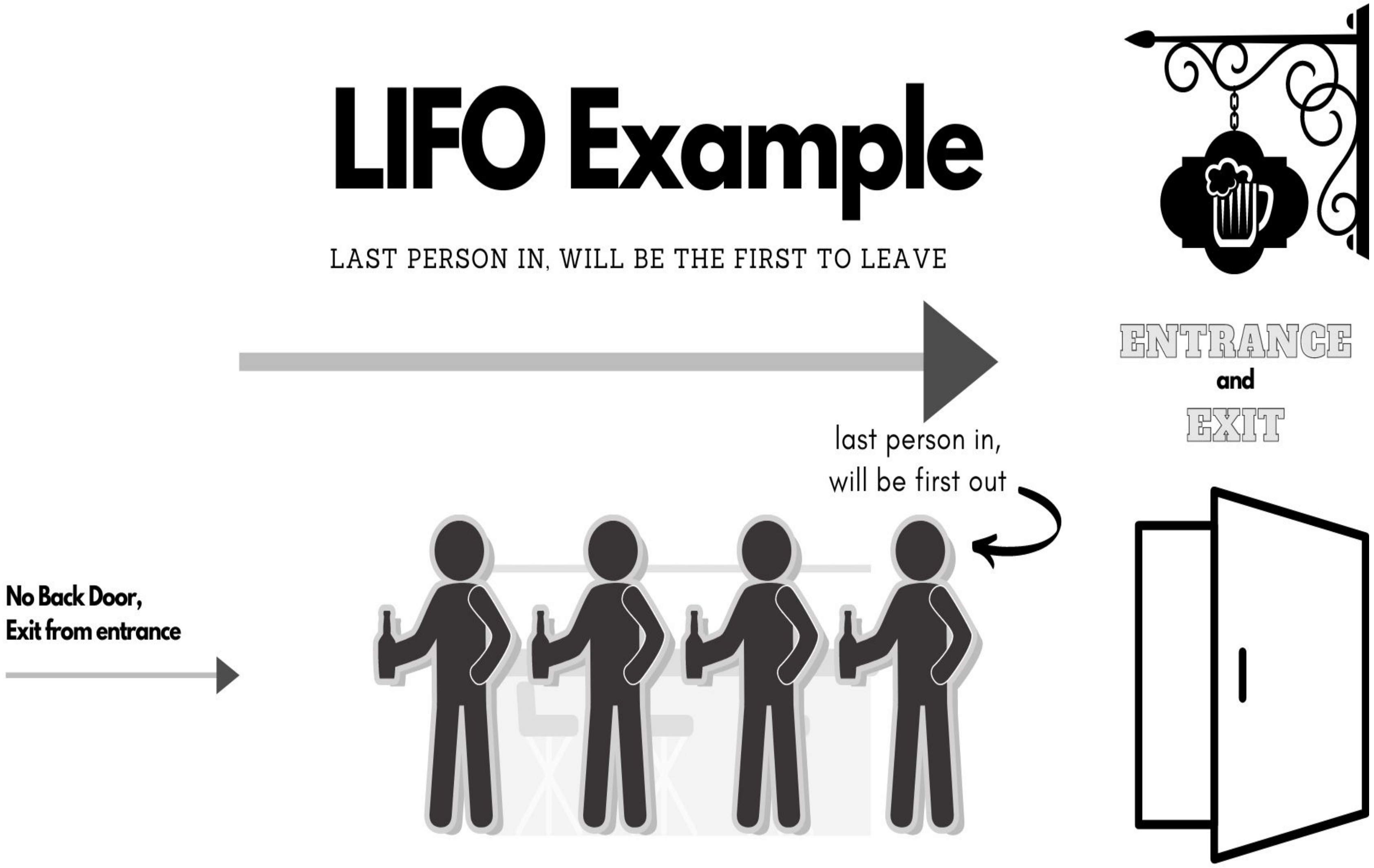
LIFO / FILO

Stack of Rings / Discs

Stack of Chairs

LIFO Example

LAST PERSON IN, WILL BE THE FIRST TO LEAVE



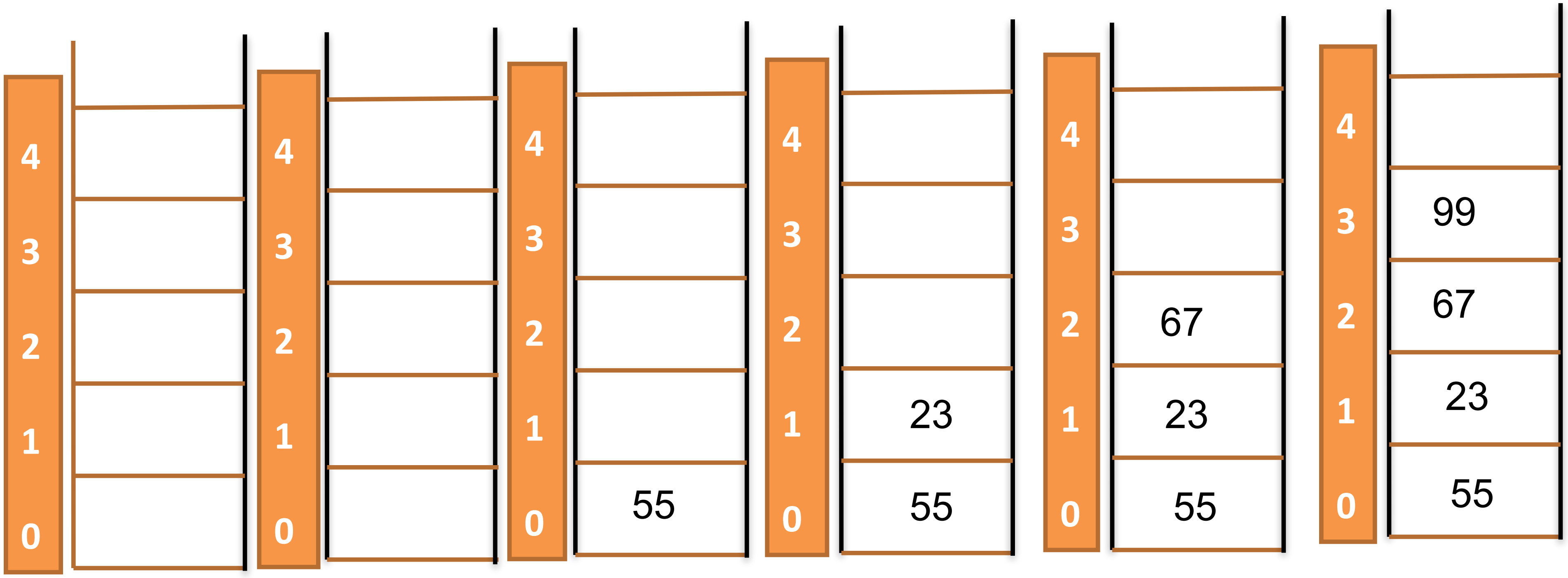
Stack Primary Operations

Two basic operations are associated with stack:

1. **“Push”** operation is used to **insert** an element into a stack.
2. **“Pop”** operation is used to **delete** an element from a stack.

Push() Example

Data to be inserted: **55, 23, 67, 99, 11**



Empty stack, size()= 5

Top=-1

55 **push(55)** **Top = 0**

23 **push(23)** **Top = 1**

67 **push(67)** **Top = 2**

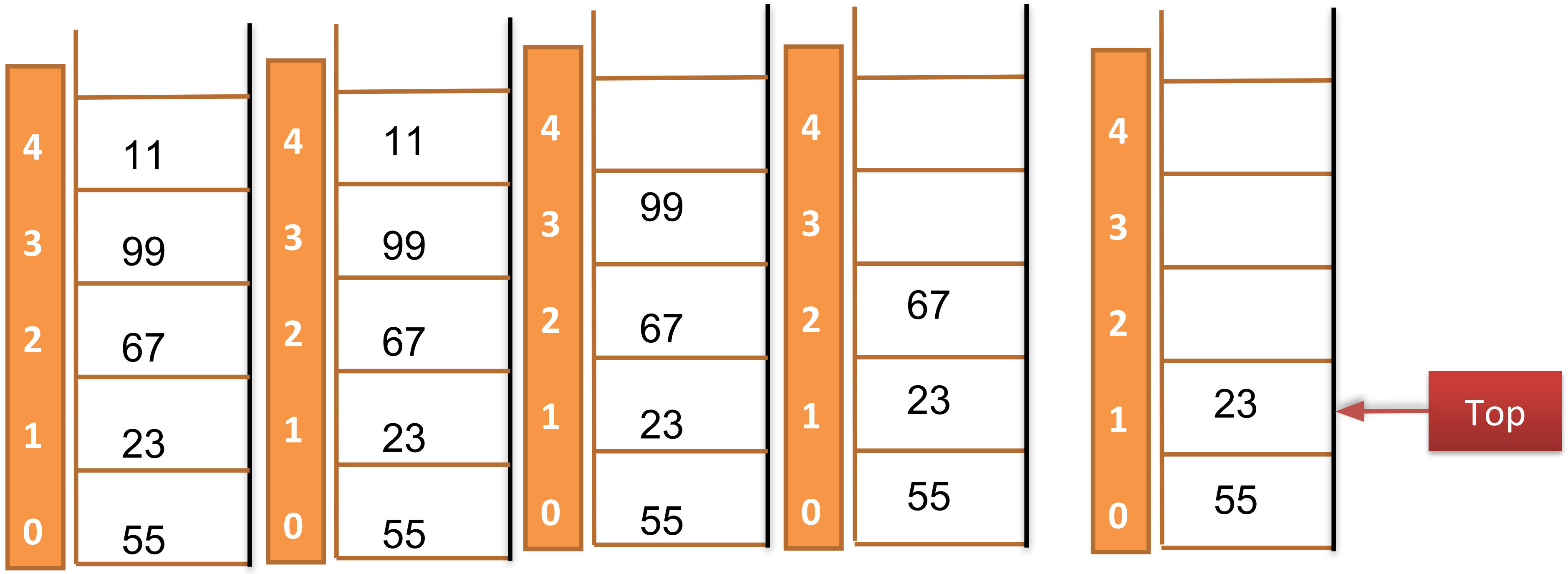
99 **push(99)** **Top = 3**

11 **push(11)** **Top = 4**

Next push(100) ?

pop() Example

Data to be deleted: **11,99,67 and 55**



stack
size()= 5

pop(11)

Top= 4

pop(99)

Top = 3

pop(67)

Top = 2

pop(55)

Top = 1
Stack [Top] = 23 ≠ 55
Operation will not be performed

Algorithms for Push and Pop Operations

For Push Operation

Push-Stack(Stack, Top, MaxSTK, Item)

Stack□ Place where to store data.

Top □In which location the data is to be inserted.

MaxSTK □ Maximum size of the stack

Item □ New item to be added.

```
1.If Top = MaxSTK then Print: Overflow and Return.   /*…Stack already filled..*/

2.Set Top := Top +1

3.Set Stack[Top] := Item

4.Return.
```

For Pop Operation

Pop-Stack(Stack, Top, Item)

Stack□ Place where to store data.

Top □ Represents from which location the data is to be removed

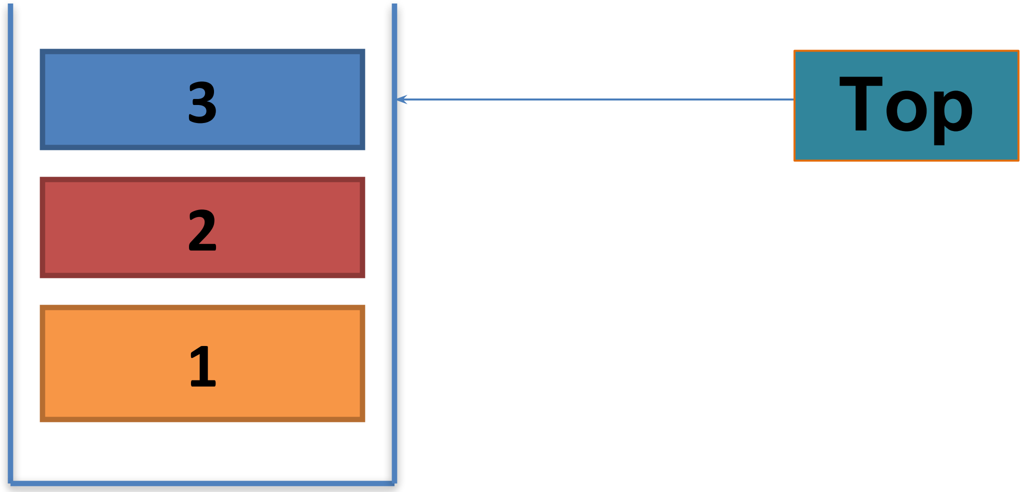
Item □ Top element is assigned to Item.

```
1.If Top = -1 then Print: Underflow and Return.        /*…Stack already Empty..*/

2.Set  Item := Stack[Top]

3.Set Top := Top - 1

4. Return.
```

Stack Secondary Operations

- Top():

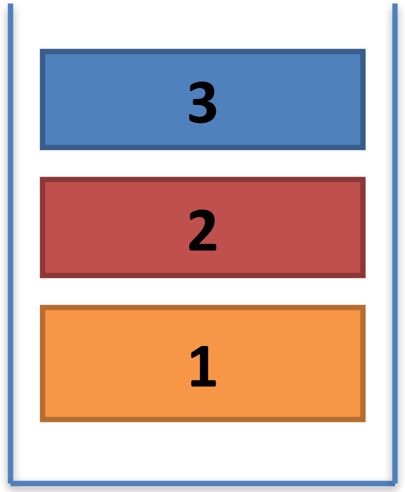
-returns the last inserted element without removing it.



Stack Secondary Operations

- Size():

-returns the size or number of elements of a stack.

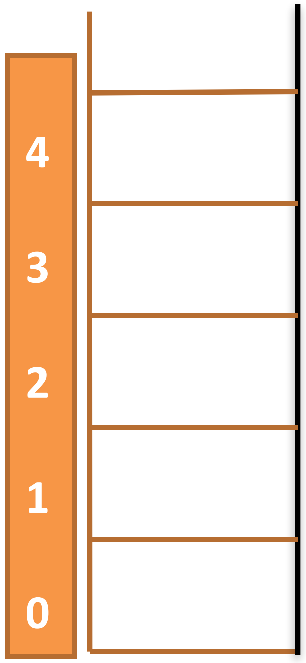


Size = 3

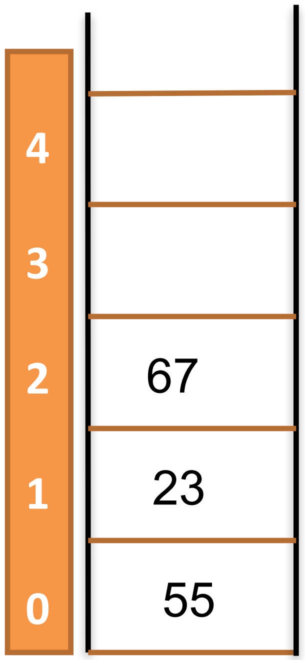
Stack Secondary Operations

- isEmpty():

- returns TRUE if the stack is Empty and FALSE otherwise.



isEmpty() ?
TRUE

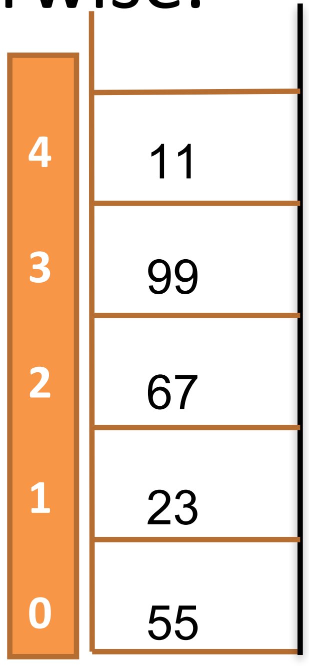


isEmpty() ?
FALSE

Stack Secondary Operations

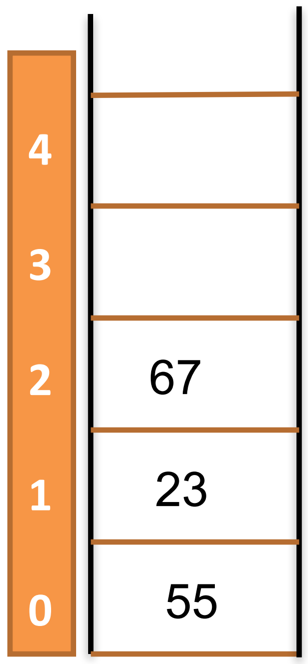
- isFull():

- returns TRUE if the stack is Full/has maximum number of elements within its size and FALSE otherwise.



□isFull() ?
TRUE

Max size = 5



□isFull() ?
FALSE

Applications of Stack

- **Arithmetic Expression Evaluation**

Calculators use a stack structure to hold values for calculation.

- **Syntax Parsing**

Many compilers use a stack for parsing the syntax of expressions before translating into low level code.

- **Solving Search Problem**

Solving a search problems, regardless of whether the approach is exhaustive or optimal, needs stack space. Example of exhaustive search methods is backtracking. Example of optimal search exploring methods is branch and bound. All of these algorithms use stacks to remember the search nodes that have been noticed but not explored yet.

- **Runtime Memory Management**

Almost all computer runtime memory environments use a special stack (the “call stack") to hold information about procedure/function calling and nesting in order to switch to the context of the called function and restore to the caller function when the calling finishes. They follow a runtime protocol between caller and callee to save arguments and return value on the stack. Stacks are an important way of supporting nested or recursive function calls.

Arithmetic Expression

- Arithmetic expression consists of operands and operators.
- Three types of arithmetic expression:

1. Infix Expression

   - Operators are placed between its two operands.

   - Example: 2 + 4, a – b / c

2. Prefix Expression(Polish Notation)

   - Operators are placed before its two operands.

   - Example: + 2 4, - a / b c

3. Postfix Expression (Reverse Polish Notation or RPN)

   - Operators are placed after its two operands.

   - Example: 2 4 +, a b c / -

Operator Precedence

- The order in which expressions are evaluated is determined by operator precedence.
- If an expression contains two or more operators with the same precedence level, the operator to the left is processed first.
- When a lower order precedence operation must be performed first, it should be surrounded by parentheses.

Operators	Precedence
( )	Highest
Unary -	Next Highest
^	Next Highest
* , / , %	Next Highest
+ , -	Lowest

Figure: Arithmetic Operator Precedence Table

Conversion of Infix Expression into Its Equivalent Prefix Expression

1. F = A + B * C /*..Infix Expression…..*/

   = A + [* B C]

   = [+ A * B C]

   = + A * B C /*..Prefix Expression…..*/

2. F = (5 + 7) * 8 - 10 /*..Infix Expression…..*/

   = [+ 5 7] * 8 - 10

   = [* + 5 7 8] - 10

   = [- * + 5 7 8 10]

   = - * + 5 7 8 10 /*..Prefix Expression…..*/

3. F = (A + B ^ D) / (E - F)+G /*..Infix Expression…..*/

   = ?

Conversion of Infix Expression into Its Equivalent Postfix Expression

```
1.   F  =   A   +   B   *  C                          /*..Infix Expression…..*/
        =   A   +  [ B   C   * ]
        =  [ A    B   C   *  + ]
        =   A    B    C   *    +                 /*..Postfix Expression…..*/

2.   F   =  ( 5  +  7 )  *  8   -  10            /*..Infix Expression…..*/
         =  [ 5   7  +]   *  8   -  10
         =  [ 5   7   +  8  * ]  -  10
         =  [ 5   7   +   8   *   10  - ]
         =    5   7   +   8   *   10  -          /*..Postfix Expression…..*/

3.  F   = ( A  +  B  ^   D ) / ( E  -  F)+G      /*..Infix Expression…..*/
        = ?
```

Conversion of Prefix Expression into Its Equivalent Infix Expression

1. F = + A * B C /*...Pretfix Expression…..*/

   = + A [B * C]

   = [A + [B * C]]

   = A + B * C /*..Infix Expression…..*/

2. F = - * + 5 7 8 10 /*..Prefix Expression…..*/

   = - * [5 + 7] 8 10

   = - [(5 + 7) * 8] 10

   = [(5 + 7) * 8] - 10]

   = (5 + 7) * 8 - 10 /*..Infix Expression…..*/

Conversion of Postfix Expression into Its Equivalent Infix Expression

1. F = A B C * + /*..Postfix Expression…..*/

   = A [B * C] +

   = [A + [B * C]]

   = A + B * C /*..Infix Expression…..*/

2. F = 5 7 + 8 * 10 - /*..Postfix Expression…..*/

   = [5 + 7] 8 * 10 -

   = [(5 + 7) * 8] 10 -

   = [[(5 + 7) * 8] - 10]

   = (5 + 7) * 8 - 10 /*..Infix Expression…..*/

Conversion of Infix Expression into Postfix Expression

Algorithm: Infix-to-Postfix (Q, P)

Here Q is an arithmetic expression in infix notation and this algorithm generates the postfix expression P using stack.

```
1.Scan the infix expression Q from left to right.
2.Initialize an empty stack.
3.Repeat step 4 to 5 until all characters in Q are scanned.
4. If the scanned character is an operand, add it to P.
5. If the scanned character is an operator Φ, then
       (a) If stack is empty, push Φ to the stack.
       (b) Otherwise repeatedly pop from stack and add to P each operator which
           has the same or higher precedence than Φ.
       (c) Push Φ to the stack.
```

6. If scanned character is a left parenthesis “(“, then push it to stack.

7. If scanned character is a right parenthesis “)”, then

   (a) Repeatedly pop from stack and add to P each operator until “(“ is encountered.

   (b) Remove “(“ from stack.

8. If all the characters are scanned and stack is not empty, then

   (a) Repeatedly pop the stack and add to P each operator until the stack is empty.

9. Exit.

Example: Q: 5 * (6 + 2) - 12 / 4 and P: ?

Infix Expression Q	Stack	Postfix Expression P
5		5
*	*	5
(	* (	5
6	* (	5, 6
+	* (+	5, 6
2	* (+	5, 6, 2
)	*	5, 6, 2, +
-	-	5, 6, 2, +, *
12	-	5, 6, 2, +, *, 12
/	- /	5, 6, 2, +, *, 12
4	- /	5, 6, 2, +, *, 12, 4
	-	5, 6, 2, +, *, 12, 4, /
		5, 6, 2, +, *, 12, 4, /, -

Postfix Expression P : 5, 6, 2, +, *, 12, 4, /, -

*

Example: Q: A * ((B + C) - D) / E and P: ?

Infix Expression Q	Stack	Postfix Expression P
A		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 -
/	/	A B C + D - *
E	/	A B C + D - * E
		A B C + D - * E /

Postfix Expression P : A B C + D - * E /

*

Do on your own

- Convert the following infix expression to postfix expression:

1. 8*(5^4+2)-6^2/(9*3)
2. A + (B * C - (D /E ^ F) * G) * H

Postfix Expression Evaluation

Algorithm: Postfix-Evaluation (P, Value)

Here P is an arithmetic expression in postfix notation and this algorithm finds the value of this expression using stack.

```
1.Scan the postfix expression P from left to right.
2.Initialize an empty stack.
3.Repeat step 4 to 5 until all characters in P are scanned.
4.  If the scanned character is an operand, push it to the stack.
5.  If the scanned character is an operator Φ, then
        (a)  Remove two top elements of stack where A is the top element and B is
             the next-to-top element.
        (b)  Evaluate T = B Φ A  and push T to the stack.
6.Pop the stack and assign the top element of the stack to Value.
7.Exit
```

Example: P : 5, 6, 2, +, *, 12, 4, /, - and Value: ?

Postfix Expression Q	Stack
5	5
6	5, 6
2	5, 6, 2
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
-	37

Value: 37

END!!!!!!!!