Chapter: 9

STACK

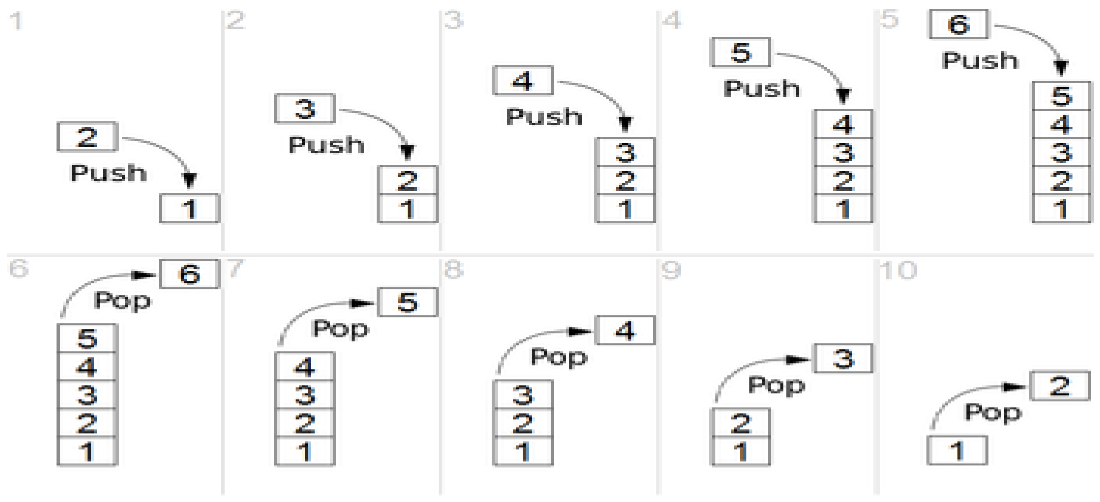
Introduction:

The term data structure refers to a data collection with well-defined operations and behaviour or properties. A stack is a linear structure implemented in LIFO (Last In First Out) manner where insertions and deletions are restricted to occur only at one end-stack's top. LIFO means element last inserted would be the first one to be deleted. The stack is also a dynamic data structure as it can grow (with increase in number of elements) or shrink (with decrease in number of elements).

A stack is a linear structure implemented in LIFO (Last In First Out) manner where insertions and deletions are restricted to occur only at one end – Stack's top. LIFO means element last inserted would be the first one to be deleted. Thus, we can say that a stack is a list of data that follows these rules :

ODM EDUCATIONAL GROUP Changing your Tomorrow

- Data can only be removed from the top (prop), i.e., the element at the top of the stack. The removal of element from a stack is technically called POP operation.
- A new data element can only be added to the top of the stack (push) . The insertion of element in stack is technically called PUSH operation.



The stack is a dynamic data structure as it can grow (with increase in number of elements) or shrink (with decreases in number of elements). A static data structure, on the other hand, is the one that has fixed size.

Other Stack Terms:

There are some other terms related to stacks, such as Peek, Overflow and Underflow.

Peek : Refers to inspecting the value at the stack's top without removing it. It is also sometimes referred as inspection.

Overflow : Refer to situation (ERROR) when one tries to push an item in stack that is full. This situation occurs when the size of the stack is fixed and cannot grow further or there is no memory left to accommodate new item.

Underflow : Refers to situation (ERROR) when one tries to pop/delete an item from an empty stack. That is, stack is currently having no item and still one tries to pop an item.

ODM EDUCATIONAL GROUP Changing your Tomorrow

Consider some examples illustrating stack:

Example:

Given a Bounded Stack of capacity 4 which is initially empty, draw pictures of the stack after each of the following steps. Initially the Stack is empty.

(i) Stack is empty
(ii) Push 'a'
(iii) Push 'b'
(iv) Push 'e'
(v) Pop
(vi) Push 'd'
(vii) Pop
(viii) Push 'e'
(ix) Push 'r'
(x) Push
(xi) Pop

(xii) Pop

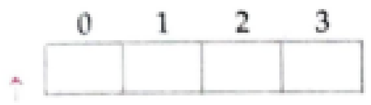
(xiii) Pop

(xiv) Pop

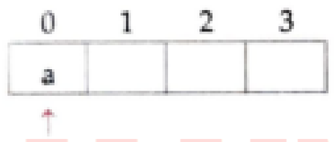
(xv) Pop

Solution

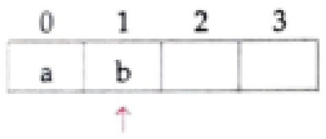
(i) Stack is empty (top = None)



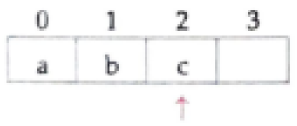
(ii) Push 'a' top = 0



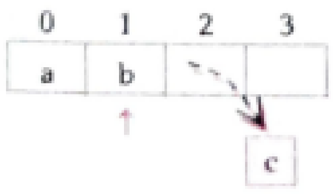
(iii) Push 'b' top = 1



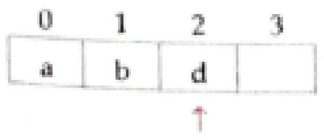
(iv) Push 'c' top =2



(v) Pop top = 1

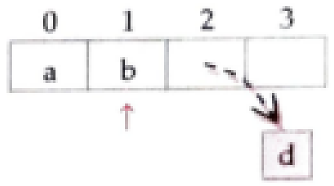


(vi) Push 'd' top = 2

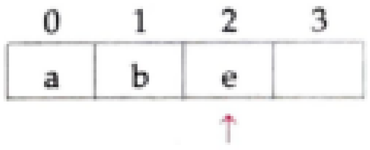


EDUCATIONAL GROUP
Changing your Tomorrow

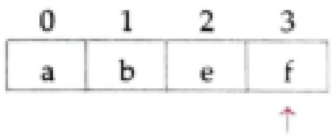
(vii) Pop top = 1



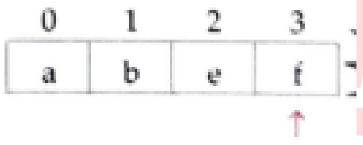
(viii) Push 'e' top = 2



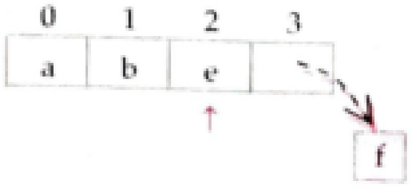
(ix) Push 'f' top = 3



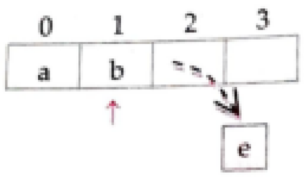
(x) Push 'g' top = 3



(xi) Pop top = 2

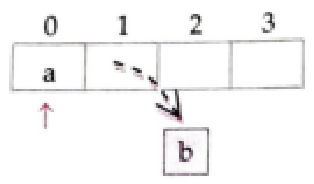


(xii) Pop top = 1.

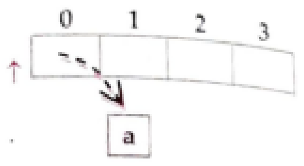


(xiii) Pop top = 0

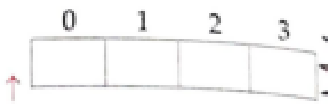
ODM EDUCATIONAL GROUP
...ging your Tomorrow



(xiv) Pop top = None



(xv) Pop top = None



UNDERFLOW

Implementing Stack in Python:

In Python, you can use Lists to implement stacks. Python offers us a convenient set of method to operate lists as stacks.

For various stack operations, we can use a list say Stack and use Python code as describe below :

Peek We can use : <Stack> [top]

Where <Stack> is a list ; top is an integer having value equal to len (<Stack>) – 1

Push We can use : <Stack>. append(<item>)

Where <item> is the item being pushed in the Stack.

Pop We can use : <Stack>. Pop( )

It removes the last value from the stack and returns it.

ODM EDUCATIONAL GROUP Changing your Tomorrow

STACK IMPLEMENTATION:

```
"""Stack : Implemented as a list

top : integer having position of topmost element in stack """

def isEmpty ( stk ) :

    if stk ==[ ]:

        return True

    else :

        return False

def Push(stk,item):

    stk.append(item)

    top=len(stk)

def Pop(stk):

    if isEmpty(stk):

        return "Underflow"

    else :

        item=stk.pop( )

        if len (stk)==0:

            top=None

        else :

            top=len(stk)-1

        return item
```

ODM EDUCATIONAL GROUP Changing your Tomorrow

```
def Peek(stk):

    if is Empty(stk):

        return "Undedrflow"

    else:

        top=len(stk)-1

        return stk[top]

def Display(stk):

    if isEmpty(stk):

        print("Stack empty")

    else:

        top=len(stk)-1

        print(stk[top],"<-top")

        for a in range (top-1,-1,-1):

            print(stk[a])
```

#_main_

```
Stack = [ ]              # initially stack is empty

top=None

while True:

    print("STACK OPERATIONS")

    print("1. Push")
```

ODM EDUCATIONAL GROUP Changing your Tomorrow

```
print("2. Pop")
print (“3.Peek”)
print("4.Display stack")
print("5,Exit")
ch=int(input("Enter your choice(1–5):"))
if ch==1:
          item=int(input("Enter item:"))
          push(Stack,item)
elif ch==2:
          item=Pop(Stack)
          if item=="Underflow":
                    print("Underflow! Stack is empty!")
          else:
                    print("Popped item is",item)
elif ch==3:
          item=Peek(Stack)
          if item=="Underflow":
                    print("Underflow!Stack is empty!")
          else:
                    print("Topmost itemis",item)
```

ODM EDUCATIONAL GROUP Changing your Tomorrow

```
    elif ch ==4 :
        Display(Stack)
    elif ch == 5 :
        break
    else :
        print("Invalid choice!")
```

Sample run of the above program is as shown below:

```
STACK OPERATIONS
1. Push
2. Pop
3. Peek
4. Display  stack
5. Exit
Enter your choice (1 – 5 ) : 1
Enter item : 6
----------------------------------------
STACK OPERATIONS
1. Push
2. Pop
3. Peek
4. Display stack
5. Exit
Ener your choice (1-5) : 1
```

```
STACK OPERATIONS
1. Push
2. Pop
3.Peek
4. Display stack
5. Exit
Enter your choice (1 – 5 ) : 1
Enter item : 4
----------------------------------------
STACK OPERATIONS
1. Push
2. Pop
3. Peek
4. Display stack
5. Exit
Enter your choice (1-5) : 4
```

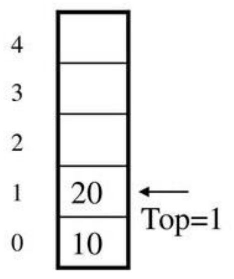
```
STACK OPERATIONS
1. Push
2.Pop
3.Peek
4. Display stack
5. Exit
Enter your choice (1 – 5 ) : 1
4 < - top
2
8
6
--------------------------------
STACK OPERATIONS
1. Push
2. Pop
```

```
Enter item : 8
----------------------------------------

STACK OPERATIONS

1. Push

2. Pop

3. Peek

4. Display stack

5. Exit

Enter your choice (1-5) : 1

Enter item : 2

------------------------------------

4 < - top
2

8

6

__________________

STACK OPERATIONS

1. Push

2. Pop

3. Peek

4. Display Stack

5. Exit

Enter your choice (1-5) : 3

Topmost item is 4

3. Peek                                   --
4. Display stack

5. Exist

Enter your choice (1-5) :5
```

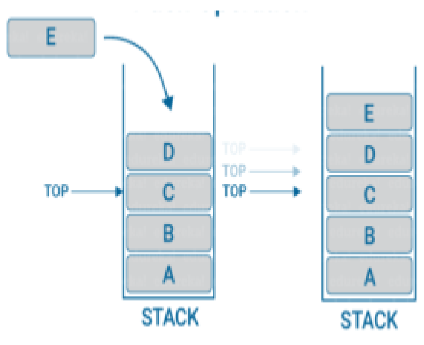
Types of Stacks:

An item stored in a stack is also called item-node sometimes. In the above implemented stack, the stack contained item-nodes containing just integers. If you want to create stack that may contain logically group information such as member details like member no, member name, age etc. For such a stack the item-node will be a list containing the member details and then this list will be entered as an item to the stack.

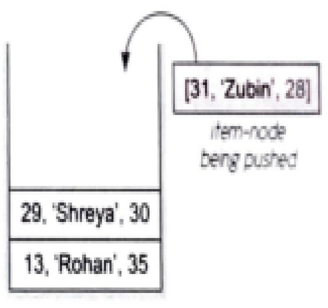
ODM EDUCATIONAL GROUP Changing your Tomorrow



(a)



(b)



(c)

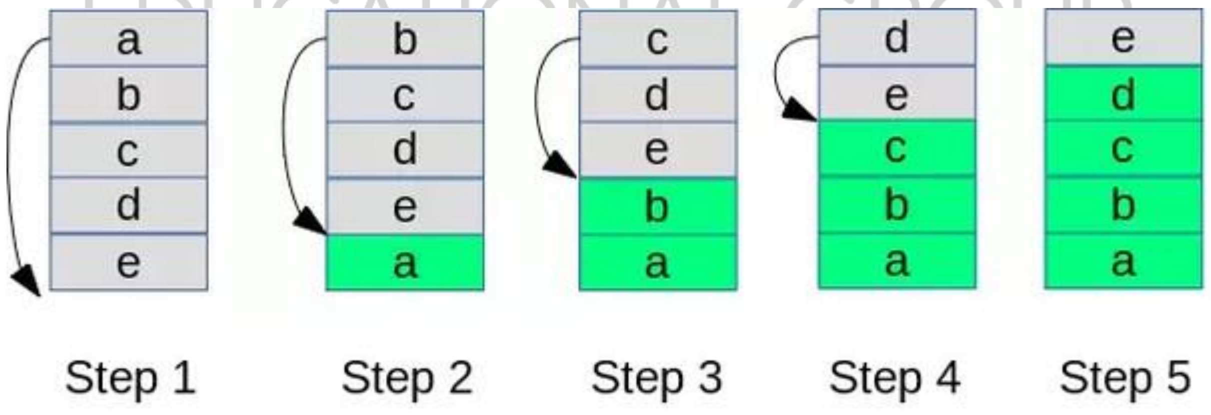
- For stack of figure(a) the stack will be implemented as Stack of integers as item-node is of integer type.
- For stack of figure(b) the stack will be implemented as stack of strings as item-node is of string type.
- For stack of figure the stack will be implemented as Stack of lists as item-node is of list type. Solved problem 20 implements such a stack.

Stack Applications:

There are several applications and uses of stacks. The stacks are basically applied where LIFO (Last In First Out) scheme is required.

Reversing a Line:

A simple example of stack application is reversal of a given line. We can accomplish this task by pushing each character on to a stack as it is read. When the line is finished. Characters are then popped off the stack and they will come off in the reverse order as shown in figure. The given line is **edcba**



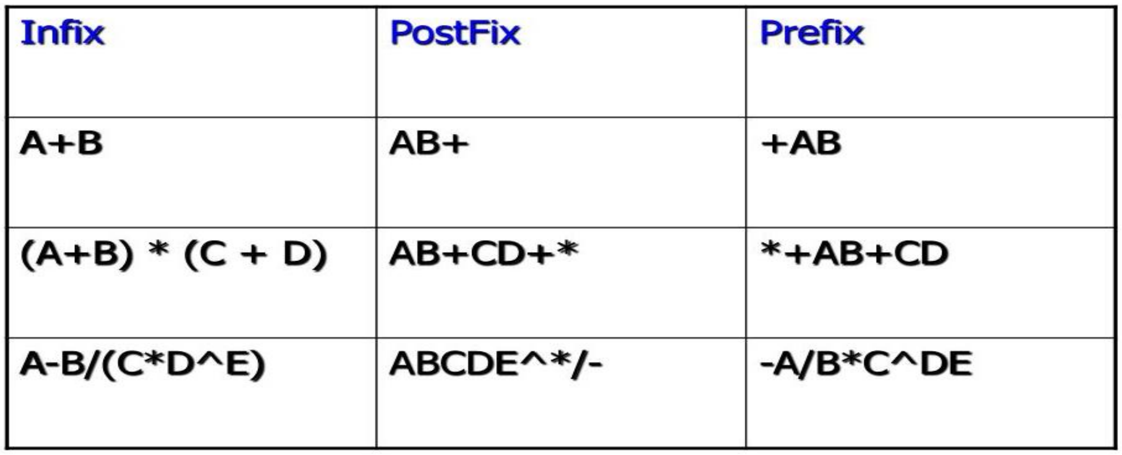
Polish String:

Another application of stacks is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. As our computer system can only understand and work on a binary language, it assumes that an arithmetic operation can take place in two operands only e.g. A + B, C x D, D/A etc. But in our usual from an arithmetic expression may consist of more than one operator and two operands e.g., $(A+B)\times C[D/(J+D)]$. These complex arithmetic operations can be converted into polish strings using stacks which then can be executed in two operands and an operator form.

Polish string, named after a polish mathematician, Jan Lukasiewicz, refers to the notation in which the operator symbol is placed either before its operands (prefix notation) or after its operands (postfix notation) in contrast to usual form where operator is placed in between the operands (infix notation).

Following table shows the three types of notations:

Expressions in infix, prefix, postfix notations:



Infix	PostFix	Prefix
A+B	AB+	+AB
(A+B) * (C + D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

Conversion of infix Expression to Postfix (Suffix) Expression:

While evaluating an infix expression, there is an evaluation order according to which

I. Brackets or Parenthesis,

II. Exponentiation,

III. Multiplication or Division

IV. Addition or Subtraction

Take place in the above specified order. The operators with the same priority (e.g., x and /) are evaluated from left to right.

To convert an infix expression into a postfix expression, this evaluation order is taken into consideration.

An infix expression may be converted into postfix from either manually or using a stack. The manual conversion requires two passes : One for inserting braces and another for conversion. However, the conversion through stack requires single pass only.

The steps to convert an infix expression into a postfix expression manually are given below:

(i) Determine the actual evaluation order by inserting braces.

(ii) Convert the expression in the innermost braces into postfix notation by putting the operator after the operands.

(iii) Repeat step (ii) until entire expression is converted into postfix notation.

Example: Convert (A + B) x C/D into postfix notation.

Solution:

Step I: Determine the actual evaluation order by putting braces

$=((A+B)xC)/D$

Step II: Converting expressions into innermost braces

$=((AB+)xC)/D=(AB+Cx)/D=AB+C\times D/$

Example: Convert $((A+B)*C/D+E\wedge F)/G$ into postfix notation.

Solution : The evaluation order of given expression will be

$=((((A+B)*C)/D)+(E\wedge F))/G$

Converting expressions in the braces, we get

$=((((AB+)*C)/D)+(EF\wedge))/G$

$=(((AB+C*)/D)+EF\wedge)/G$

$=((AB+C*D/)+EF\wedge)/G=(AB+C*D/EF\wedge+)/G$

$=AB+C*D/EF\wedge G/$

Example: **Give postfix form of the following expression** $A*(B+(C+D)*(E+F)/G)*H$

Solution: Evaluation order is

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

Converting expressions in the braces, we get

$=(A*(B+[(CD+)*(EF+)]/G))*H=A*(B+(CD+EF+*)/G)$

$=A*(B+(CD+EF+*G/))*H=(A*(BCD+EF+*G/+))*H$

$=(ABCD+EF+*G/+*)*H=ABCD+EF+*G/+*H*$

Example: **Give postfix form for** $A+[(B+C)+(D+E)*F]/G$

Solution: Evaluation order is: $A+[\{|(B+C)+((D+E)*F)\}/G]$

Converting expressions in braces, we get

$=A+[\{(BC+)+(DE+)*F\}/G]=A+[\{(BC+)+(DE+F*)\}/G]$

$=A+[\{BC+DE+F*+\}/G]=A+[BC+DE+F*+G/]$

$=ABC+DE+F*+G/+$

Example: **Give postfix form of expression for the following:** **NOT A OR NOT B NOT C**

Solution: The order of evaluation will be

$$((\text{NOT}A)\ \text{OR}((\text{NOT }B)\ \text{AND}\ (\text{NOT }C)))$$ (As priority order is NOT, AND, OR)

$$=((\text{AND})\ \text{OR}((B\ \text{NOT})\ \text{AND}(C\ \text{NOT})))$$

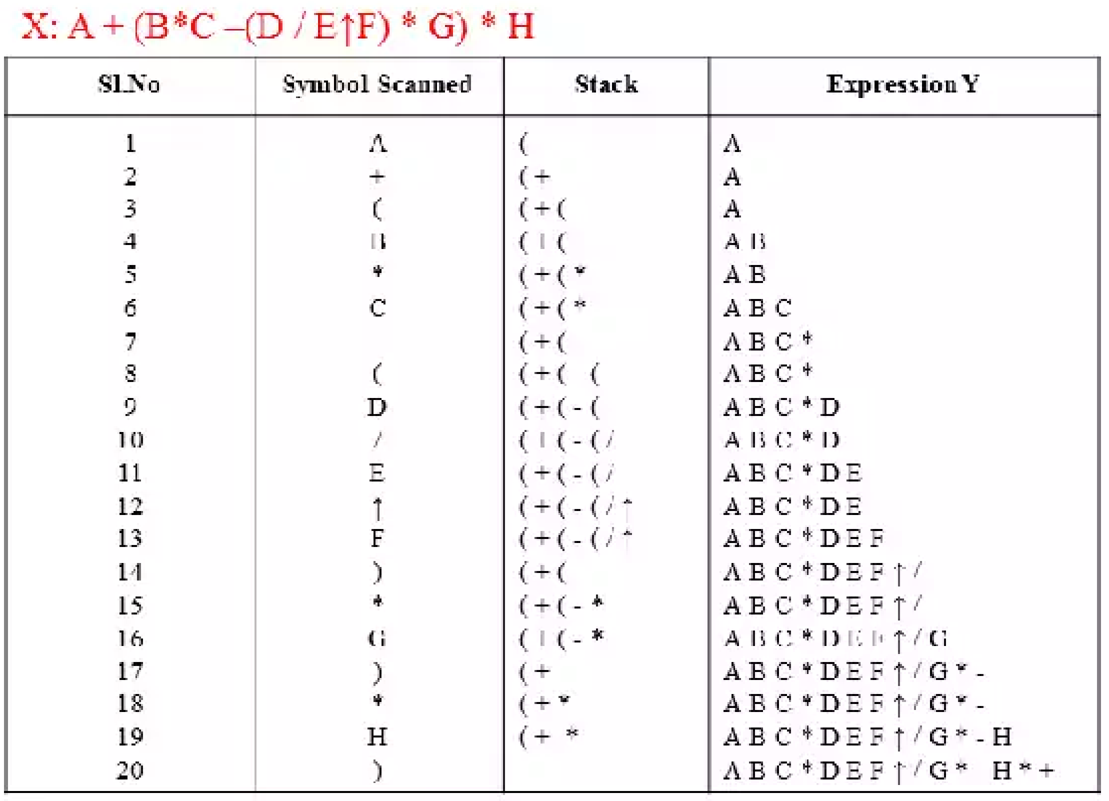
$$=((A\ \text{NOT})\ \text{OR}\ ((B\ \text{NOT}\ C\ \text{NOT}\ \text{AND})))$$

$$=A\ \text{NOT}\ B\ \text{NOT}\ C\ \text{NOT}\ \text{AND}\ \text{OR}$$

While converting from infix to prefix form, operators are put before the operands. Reverse conversion procedure is like that of infix to postfix conversion.

Example: Convert into postfix form showing stack status after every step in tabular form.

$X: A+(B+C-(D/E\hat{}F)*G)*H$



X: A + (B*C –(D / E↑F) * G) * H

Sl.No	Symbol Scanned	Stack	Expression Y
1	A	(	A
2	+	(+	A
3	(	(+ (	A
4	B	(+ (	A B
5	*	(+ (*	A B
6	C	(+ (*	A B C
7		(+ (	A B C *
8	(	(+ ((	A B C *
9	D	(+ (- (	A B C * D
10	/	(+ (- (/	A B C * D
11	E	(+ (- (/	A B C * D E
12	↑	(+ (- (/ ↑	A B C * D E
13	F	(+ (- (/ ↑	A B C * D E F
14	)	(+ (	A B C * D E F ↑ /
15	*	(+ (- *	A B C * D E F ↑ /
16	G	(+ (- *	A B C * D E F ↑ / G
17	)	(+	A B C * D E F ↑ / G * -
18	*	(+ *	A B C * D E F ↑ / G * -
19	H	(+ *	A B C * D E F ↑ / G * - H
20	)		A B C * D E F ↑ / G * H * +

Evaluation of a postfix Expression using stack:

As postfix expression is without parenthesis and can be evaluated as two operands and an operator at a time, this becomes easier for the compiler and the computer to handle. Evaluation rule of a postfix expression states:

- While reading the expression from left to right, push the element in the stack if it is an operand:
- Pop the two operands from the stack if the element is a binary operator. In case of NOT operator. Pop one operand from the stack and then evaluate it (two operands and in operator).
- Push back the result of the evaluation, repeat it till the end of the expression.

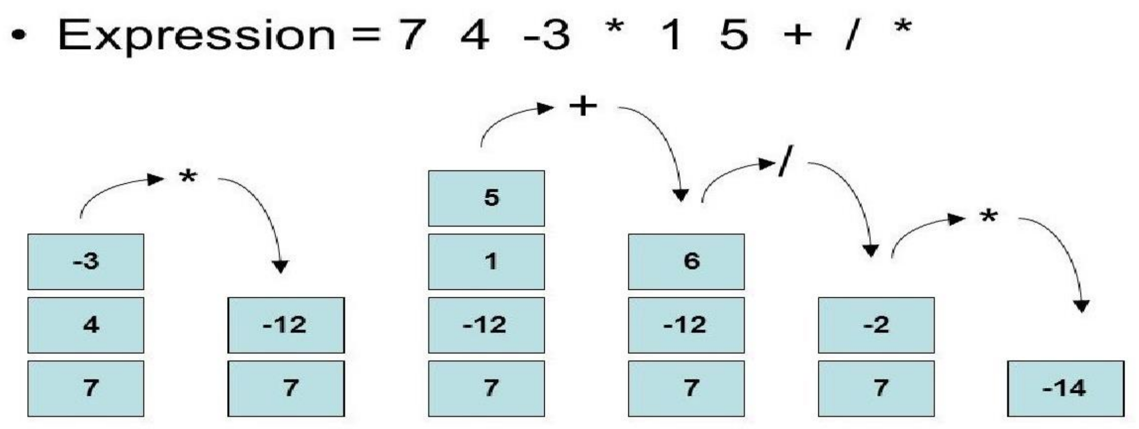
For a binary operator, two operands one popped from stack and for a unary operator one operand is popped. Then, the result is calculated using operand(s) and the operator and pushed back into the stack.

Algorithm: Evaluation of Postfix Expression:

```
Reading of expression takes place from left to right’’’’

1.  Read the next element ‘’’first element for the first time’’’
2.  If element is operand, then
    Push the element in the stack
3.  If element is operator, then
    {
4.          Pop two operands from the stack
                ‘’’ POP one operand in case of unary operator’ ‘‘
5.          Evaluate the expression formed by the two operands and the operator
6.          Push the result of the expression in the stack end
    }
7.  If  no-more-elements, then
            POP the result
    Else
            Go to step 1
8.  END
```

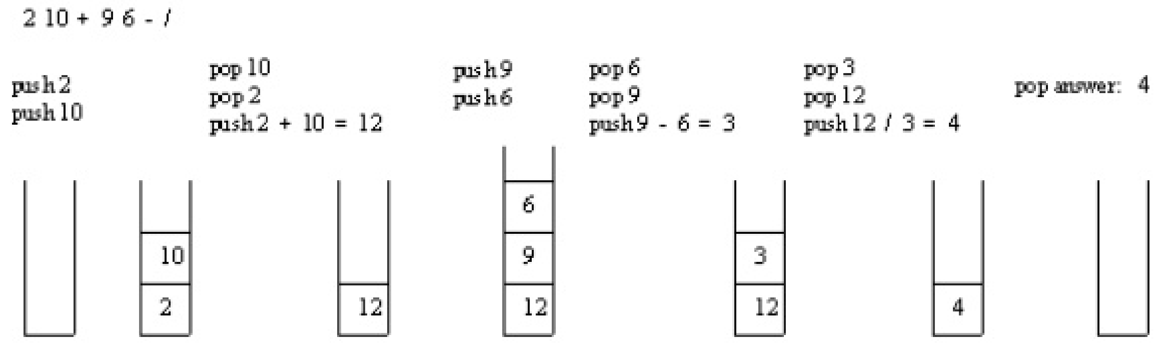
ODM EDUCATIONAL GROUP Changing your Tomorrow

Example: Evaluate the profit expression 7 4 -3 * 1 5 + / *

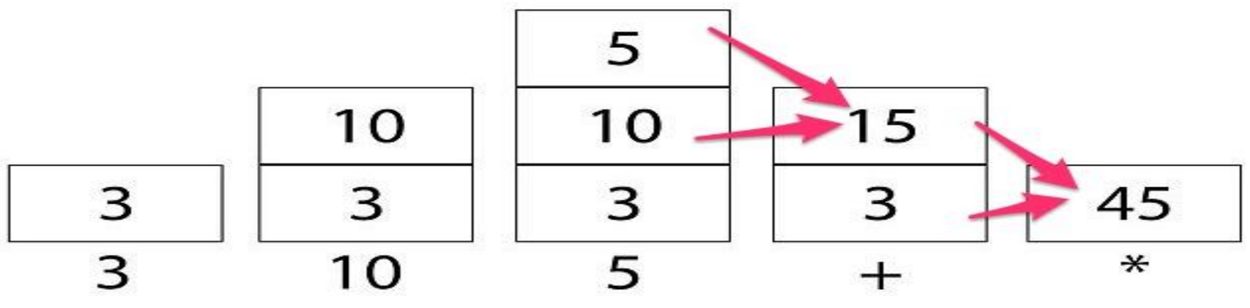


Example : **Evaluate the profit expression 2 10 + 9 6 - /**

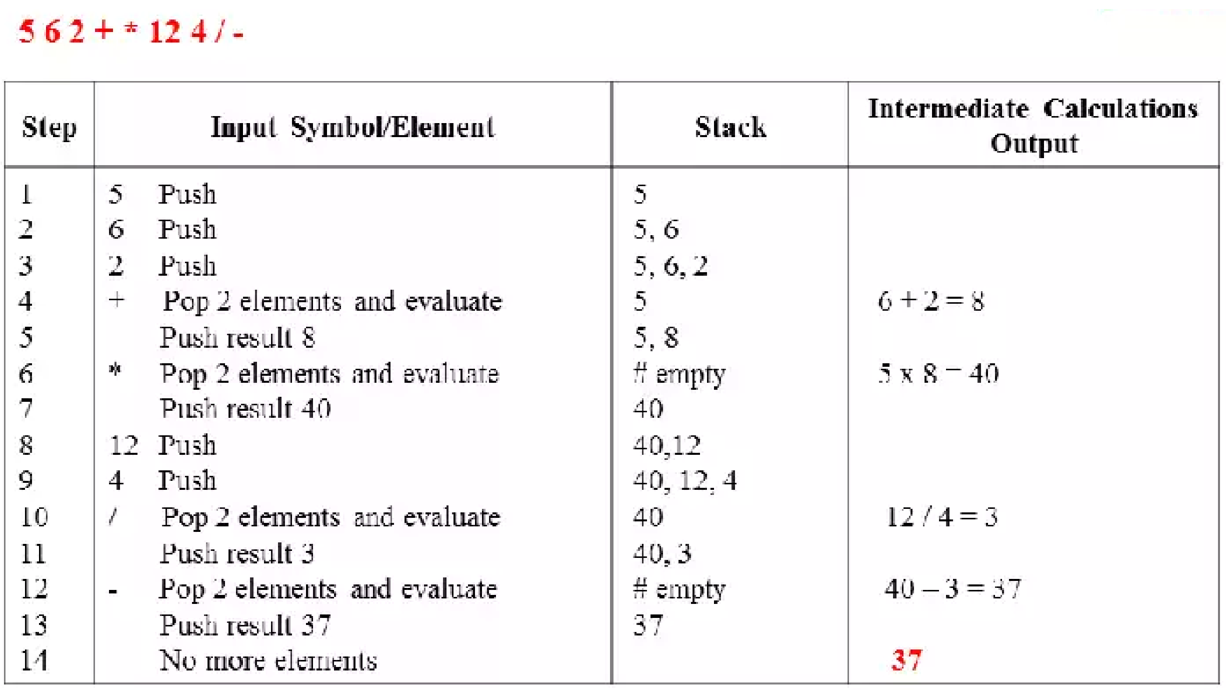
ODM



Example : **Evaluate the profit expression 3 10 5 + ***



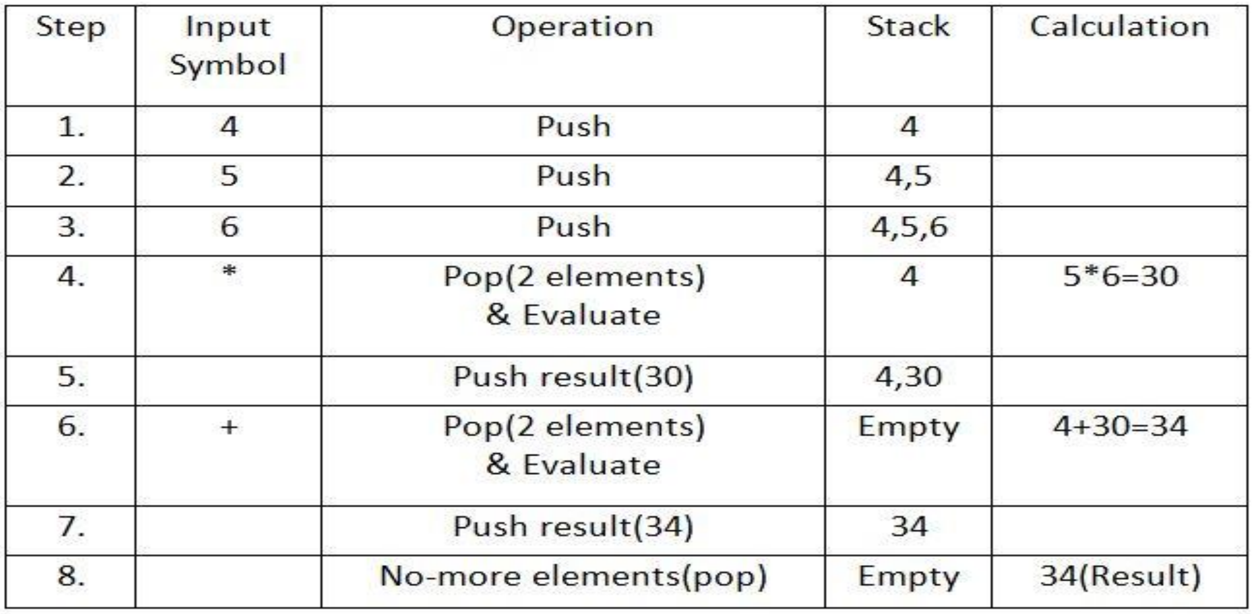
Example: Evaluate the expression 562 * 12 4/ - in tabular form showing stack status after every step.



5 6 2 + * 12 4 / -

Step	Input Symbol/Element	Stack	Intermediate Calculations Output
1	5 Push	5	
2	6 Push	5, 6	
3	2 Push	5, 6, 2	
4	+ Pop 2 elements and evaluate	5	6 + 2 = 8
5	Push result 8	5, 8	
6	* Pop 2 elements and evaluate	# empty	5 x 8 = 40
7	Push result 40	40	
8	12 Push	40,12	
9	4 Push	40, 12, 4	
10	/ Pop 2 elements and evaluate	40	12 / 4 = 3
11	Push result 3	40, 3	
12	- Pop 2 elements and evaluate	# empty	40 – 3 = 37
13	Push result 37	37	
14	No more elements		37

Example : Evaluate the expression 4 5 6 * + in tabular form showing stack status after every step.



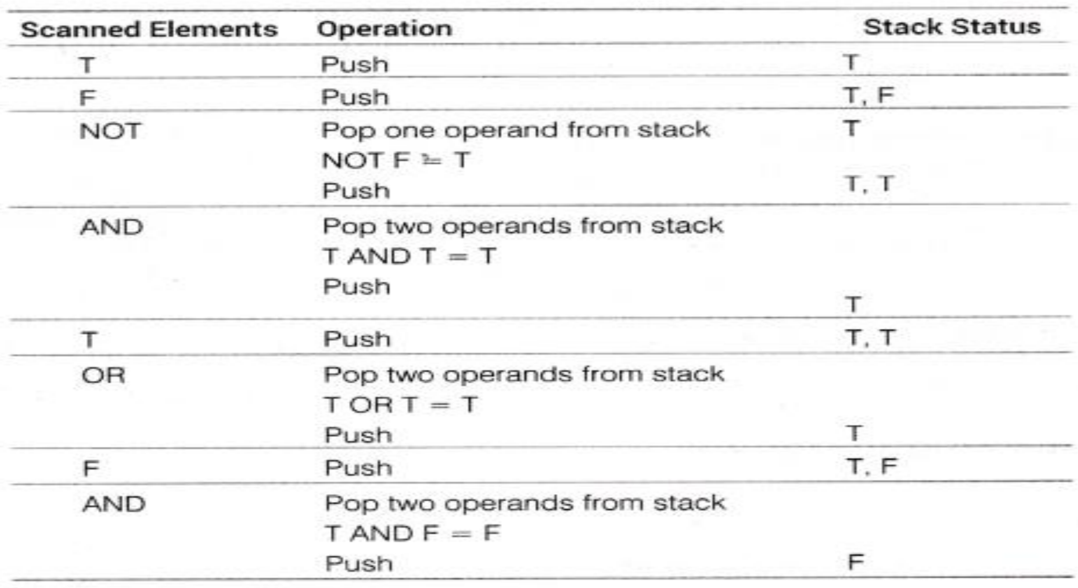
Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)

Example : Evaluate the expression True False NOT AND True True AND OR in tabular form showing stack status after every step.

Step	Input Symbol	Action Taken	Stack status	Output
1.	True	Push	True (↑ - top) ↑	
2.	False	Push	True, False ↑	
3.	NOT	Pop (1 element) Push result (True)	True ↑ True, True ↑	NOT False = True
4.	AND	Pop (2 element) Push result (True)	↑ (empty stack) True ↑	True AND True = True
5.	True	Push	True, True ↑	
6.	True	Push	True, True, True ↑	
7.	AND	Pop (2 element) Push result (True)	True ↑ True, True ↑	True AND True = True
8.	OR	Pop (2 element) Push result (True)	↑ (empty stack) True ↑	True OR True = True
			Ans = True	

Example : Evaluate the expression T F NOT AND T OR F AND in tabular form showing stack status after every step.

EDUCATIONAL GROUP



Scanned Elements	Operation	Stack Status
T	Push	T
F	Push	T, F
NOT	Pop one operand from stack NOT F = T Push	T T, T
AND	Pop two operands from stack T AND T = T Push	T
T	Push	T, T
OR	Pop two operands from stack T OR T = T Push	T
F	Push	T, F
AND	Pop two operands from stack T AND F = F Push	F