

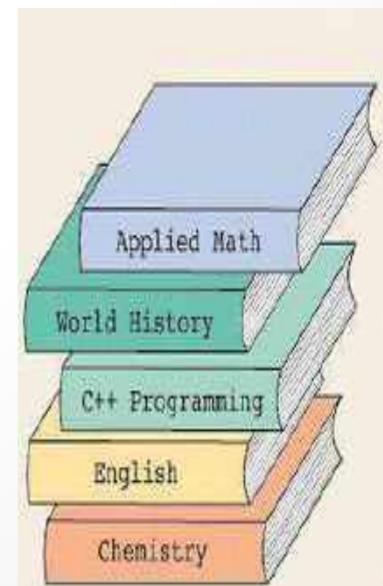
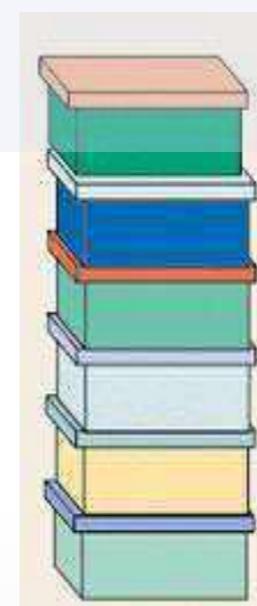
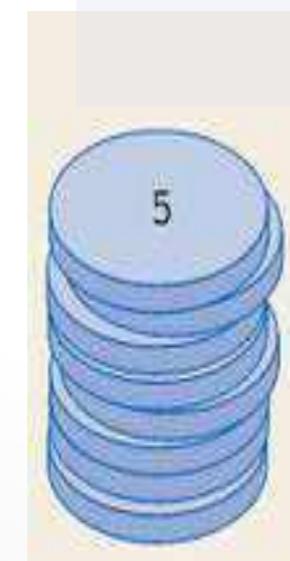
CONCEPT OF STACKS

Definition :

“ Stack is data structure in which addition and removal of an element is allowed at the same end is called as top of stack.”

- Stack is also called as Last In First Out(LIFO) list.
- It means element which get added at last will be removed first.

e.g.



Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

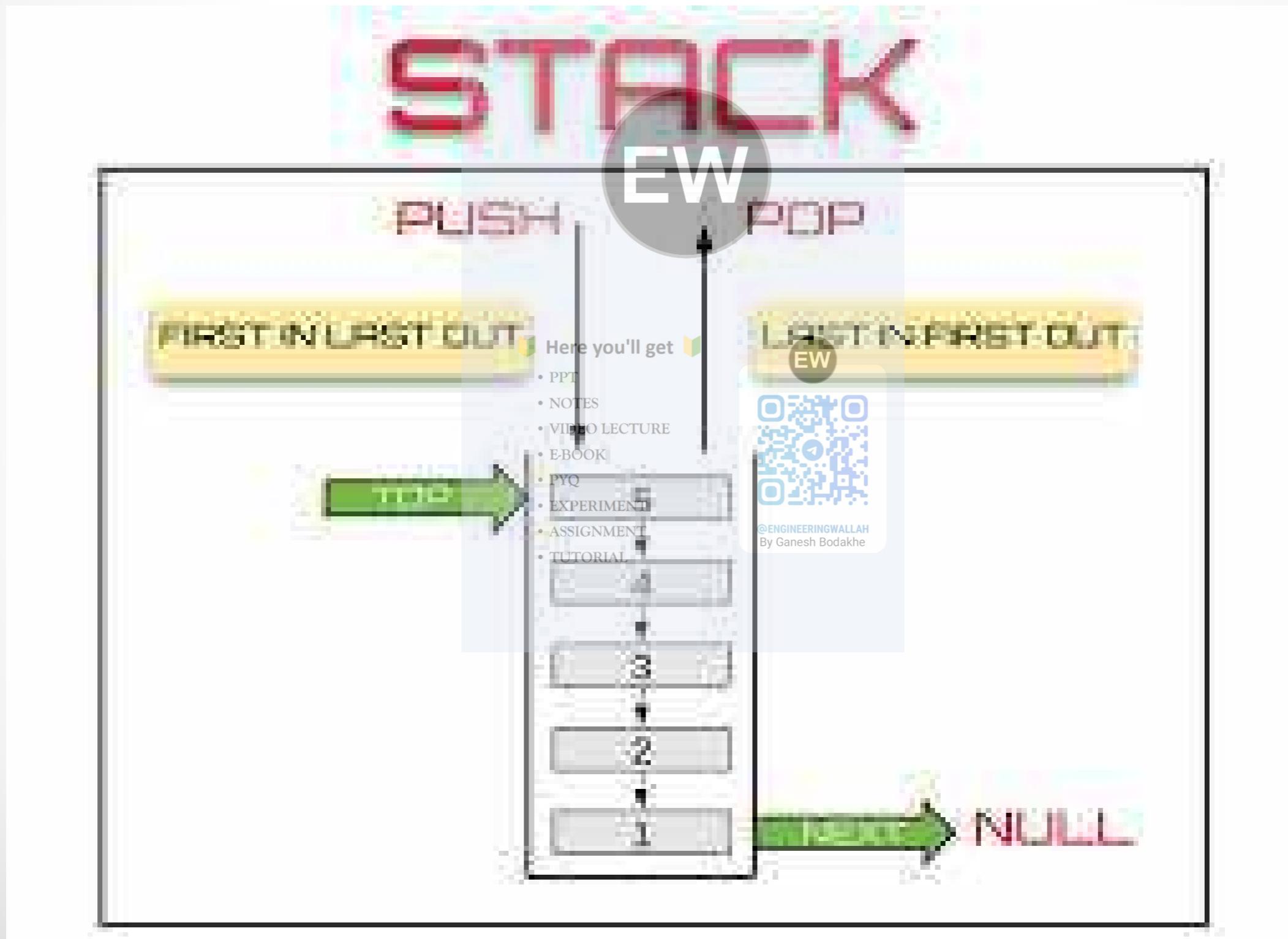


@ENGINEERINGWALLAH
By Ganesh Bodakhe



CONCEPT OF STACKS

Example :

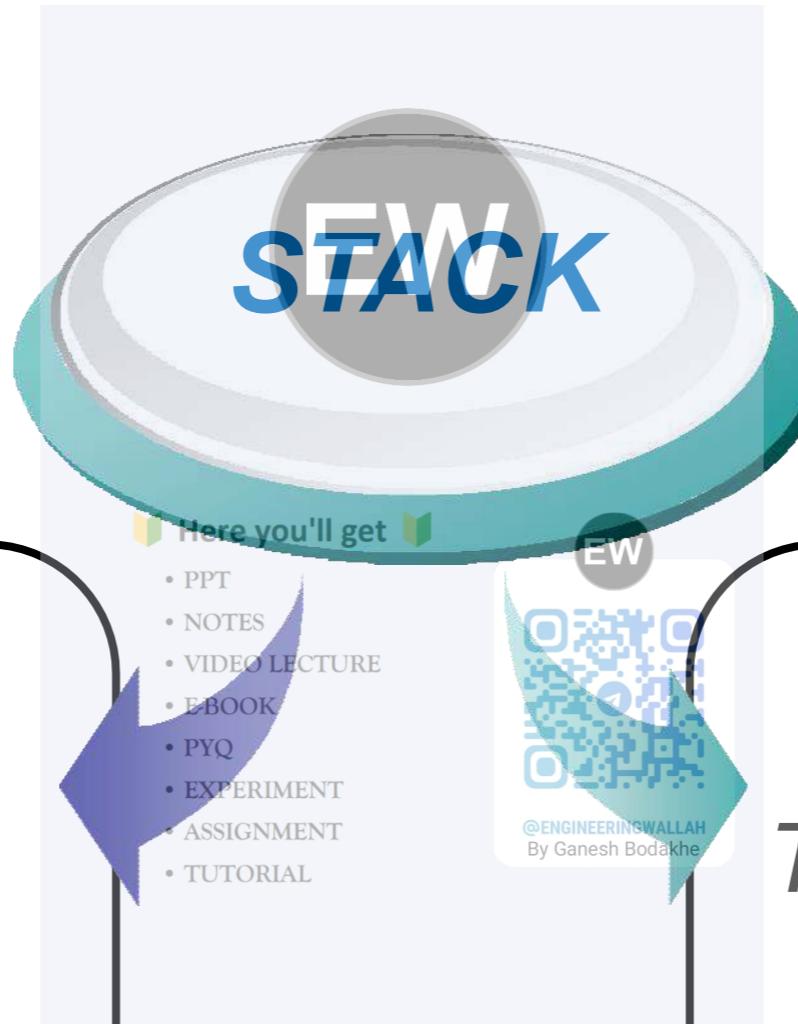


Components of Stack

- **Top** is a variable which refers to last position in stack.
- **Element** is component which has data.
- **MaxStack** is variable that describes maximum number of elements in a stack.



Main Operation



PUSH

Add data to element in stack.

POP

Take data from element in stack

Kinds of Operation

- *Stack Operation in array form*
 - Here you'll get:
 - PPT
 - NOTES
 - VIDEO LECTURE
 - E-BOOK
 - PYQ
 - EXPERIMENT
 - ASSIGNMENT
 - TUTORIAL
- *Stack Operation in Linked list form*

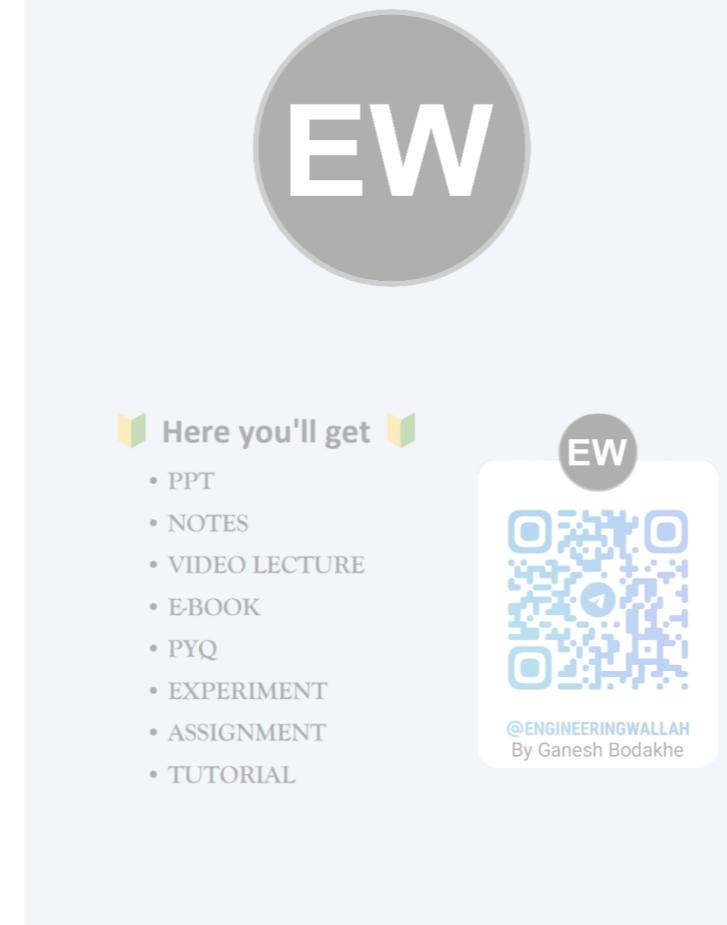


STACK AS AN ADT

Stack is an abstract data type which is defined by the following structure and operation.

Stack operation :

1. createstack()
2. Push()
3. Pop()
4. Peek()
5. IsEmpty()
6. IsFull()
7. .Size()



STACK AS AN ADT

Createstack - it create new empty stack.

push() – Pushing (storing) an element on the stack.

pop() – Removing an element from the stack.

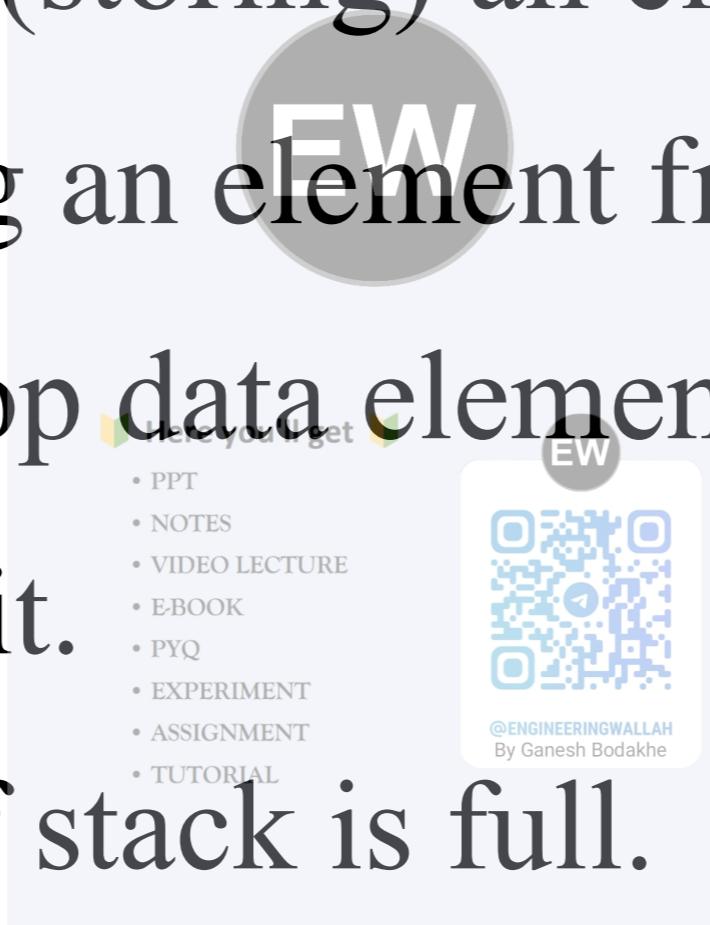
peek() – get the top data element of the stack,

without removing it.

isFull() – check if stack is full.

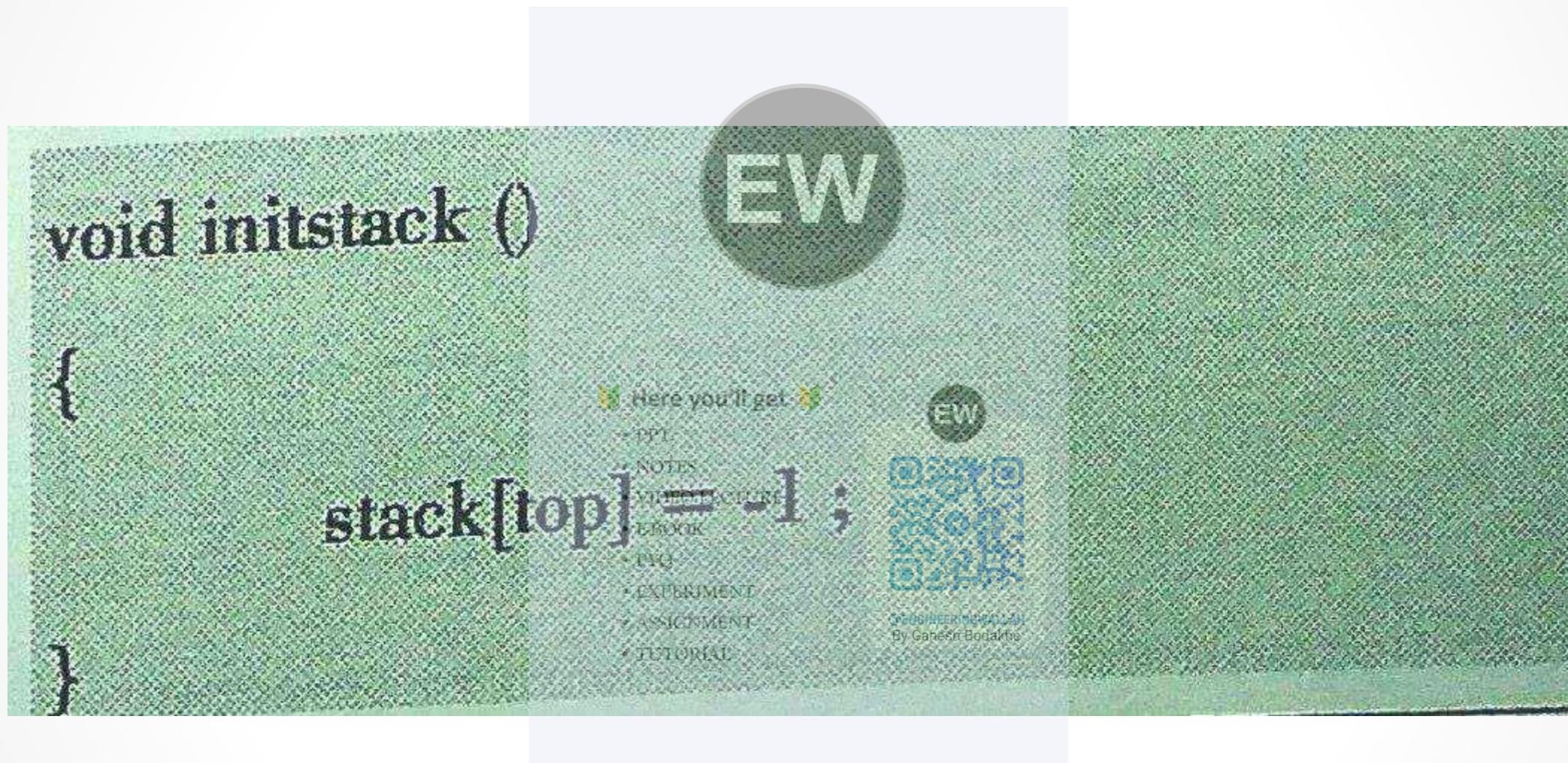
isEmpty() – check if stack is empty.

Size() – return the number of item in the stack.



STACK AS AN ADT

1. Initializing stack :

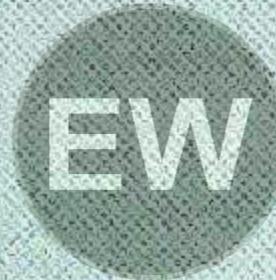


STACK AS AN ADT

2. IsFull() stack :

check stack is full or not?

```
int is_full()
{
    if(top == SIZE-1)
        return(1);
    else
        return(0);
}
```



Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- EBOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



@EASYENGINEERING
By Ganesh Bodakhe

STACK AS AN ADT

3. IsEmpty() stack :

check stack is empty or not?

```
int is_empty()
{
    if(top === -1)
        return(1);
    else
        return(0);
}
```



Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- EBOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

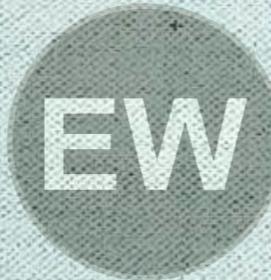


@ENGINEERINGWALLAH
By Ganesh Bodakhe

STACK AS AN ADT

4. Push() stack : add element in stack

```
void push()
{
    if(is_full() == 1)
    {
        printf("\n\tSTACK is over flow");
    }
    else
    {
        printf(" Enter an element to add in the stack :");
        scanf("%d",&ele);
        top++;
        stack[top] = ele;
    }
}
```



Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

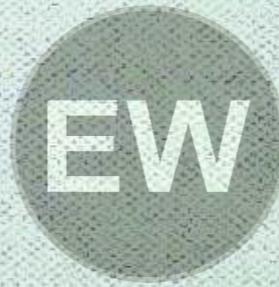


@ENGINEERINGWALLAH
By Ganesh Bodakhe

STACK AS AN ADT

5. pop() stack : remove element from stack

```
void pop()
{
    if(is_empty() == 1)
    {
        printf("\n\nSTACK is under flow");
    }
    else
    {
        printf("\n Element popped : %d",stack[top]);
        top--;
    }
}
```



- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



@ENGINEERINGWALLAH
By Ganesh Bodakhe

STACK AS AN ADT

6. display() stack :displaying stack

```
void display()
{
    int i;
    if(is_empty() == 1)
    {
        printf("\n\nSTACK is under flow");
    }
    else
    {
        printf("\n Stack elements : ");
        for(i = top;i>=0;i--)
        {
            printf(" %d ",stack[i]);
        }
    }
}
```



Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT



@ENGINEERINGWALLAH
By Ganesh Bodakhe

ALGORITHM TO IMPLEMENT STACK USING ARRAY

Step 1 : start

Step 2 : Display Menu : 1. push 2. pop 3. display
4. exit.

Step 3 : read choice

Step 4 : if choice 1 then call push ()
if choice 2 then call pop ()
if choice 3 then call display ()
if choice 4 then call exit ()
default : Invalid choice

Step 5 : read choice again

Step 6 : If choice between 1- 3
repeat step 4 else stop

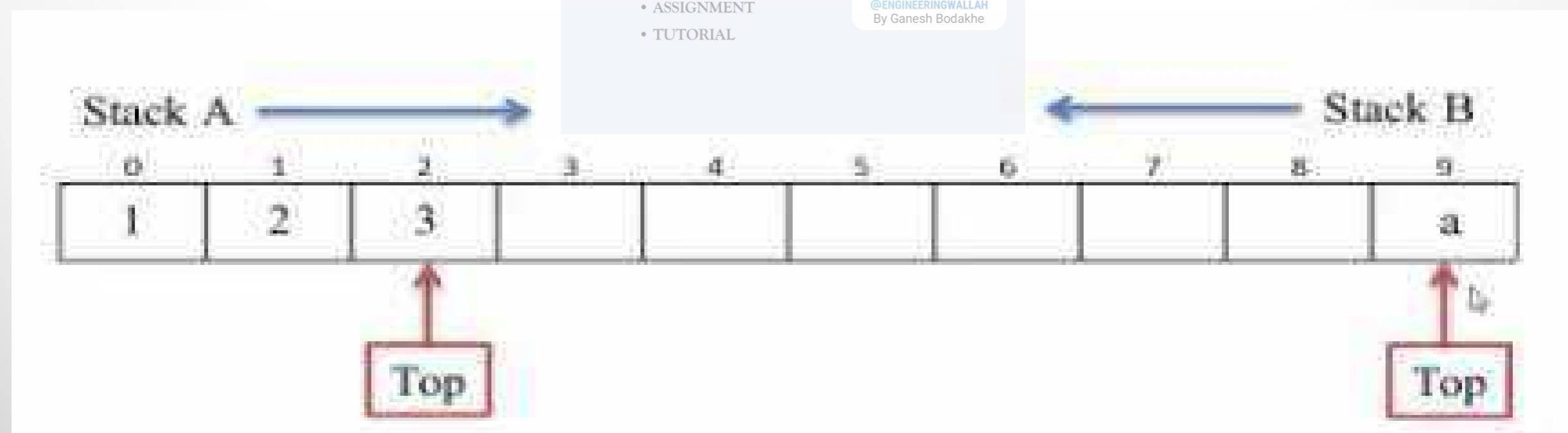
MULTIPLE STACKS

“When a stack is created using single array, we can not able to store large amount of data, thus this problem is rectified using more than one stack in the same array of sufficient array. This technique is called as **Multiple Stack**”



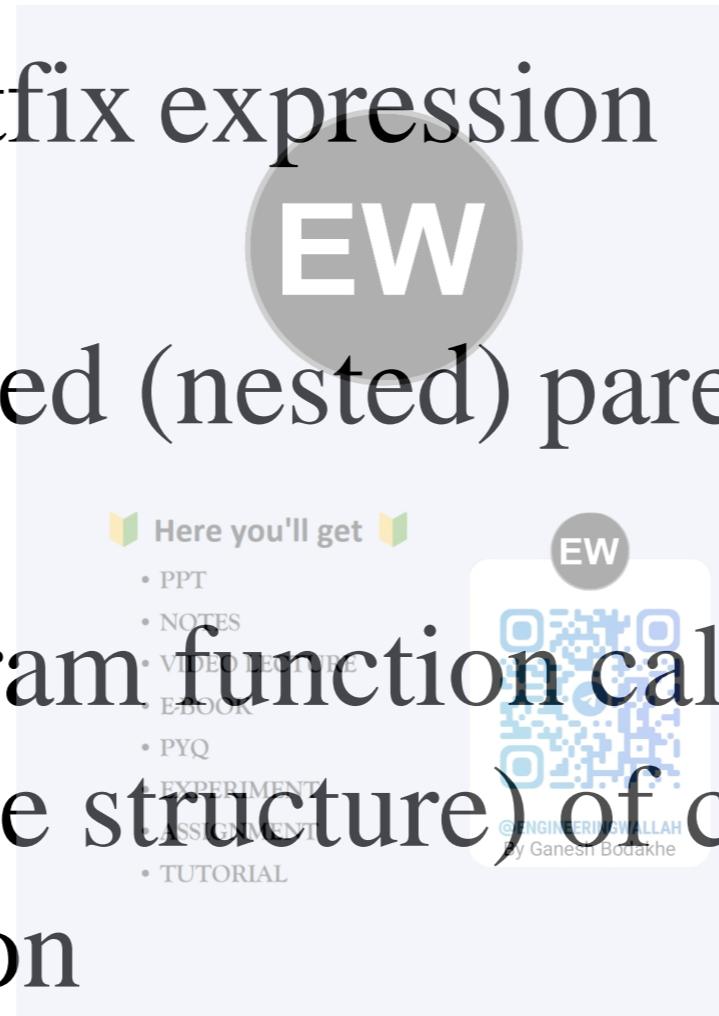
MULTIPLE STACKS

Example : When an array of $\text{STACK}[n]$ is used to represent two stacks, say Stack A and Stack B. Then the value of n is such that the combined size of both the $\text{Stack}[A]$ and $\text{Stack}[B]$ will never exceed n. Stack[A] will grow from left to right, whereas Stack[B] will grow in opposite direction i.e. right to left.



APPLICATION OF STACKS

- ❖ Convert infix expression to postfix and prefix expressions
- ❖ Evaluate the postfix expression
- ❖ Reverse a string
- ❖ Check well-formed (nested) parenthesis
- ❖ Reverse a string
- ❖ Process subprogram function calls
- ❖ Parse (analyze the structure) of computer programs
- ❖ Simulate recursion
- ❖ In computations like decimal to binary conversion
- ❖ In Backtracking algorithms (often used in optimizations and in games)



REVERSING OF STACKS

Algorithms :

Step 1 : start

Step 2 : accept string

Step 3 : insert string into character by character

using push method

Step 4 : remove character from stack one by one

and print using pop method

Step 5 : stop



REVERSING OF STACKS

```
/*program For Reverse String */
```

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
class stack
{
    char stk[MAX];
    int top;
public:
    stack()
    {   top=-1; }
```



```
void push()
{ int n,i;
    cout<<"Enter the size of string";
    cin>>n;
    if(n>MAX)
    {
        cout<<"out of size";
    } else
    { for(i=0;i<n;i++)
        cin>>stk[++top];
    }
}
```

EW

main()

```
{ int ch;
```

```
clrscr();
```

```
stack st;
```

```
while(1)
```

```
{ cout<<"\n1.push 2.reverse 3.exit\n Enter ur
```

choice"

@ENGINEERINGWALLAH

By Ganesh Bodakhe

```
cin >> ch;
switch(ch)
```

```
{ case 1: st.push();break;
case 2: st.reverse();break;
case 3: exit(0);
```

```
} }
```

}

return (0);

polish notation – Expression

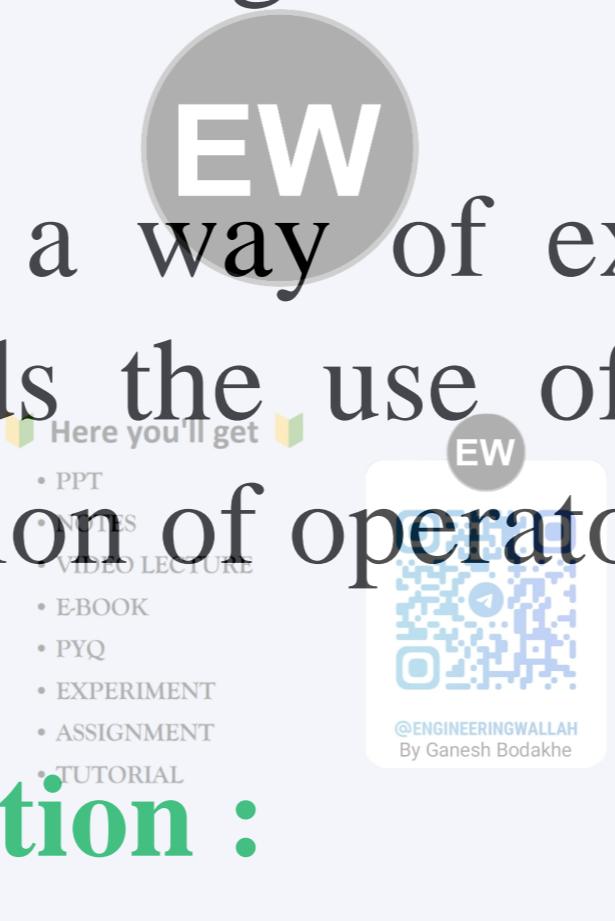
Evaluation and conversion

Notation is a way of writing arithmatic expression

Concepts : polish is a way of expressing arithmatic expression that avoids the use of brackets to define periorities for evaluation of operators.

There are three notation :

1. Infix notation
2. Prefix notation
3. Postfix notation



polish notation – Expression

Evaluation and conversion

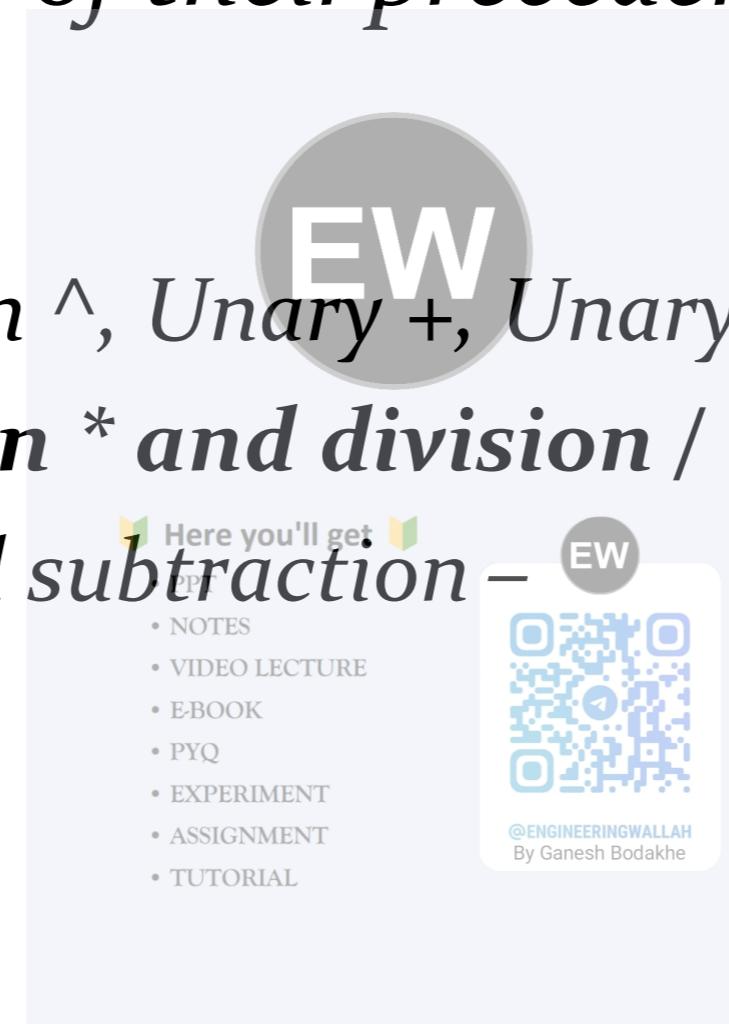
Infix	Prefix	Post fix
(operand) (operator)	(operator) (operand)	(operand) (operator)
(operand)	Here you'll get (operator) (operand)	(operator):
(A+B)*C	*+ABC	AB+C*

The example expression in various forms-
infix, prefix and postfix

- ❖ The postfix expressions can be evaluated easily hence infix expression is converted into postfix expression using stack.

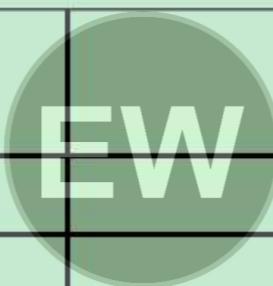
- ❖ The following operators are written in descending order of their precedence:

- ❖ Exponentiation \wedge , Unary +, Unary -, and not ~
- ❖ Multiplication * and division /
- ❖ Addition + and subtraction -
- ❖ <, £, =, 1, 3, >
- ❖ AND
- ❖ OR



The Operators and priorities

Operator	Priority
Arithmetic, Boolean and relational	
\wedge , Unary +, Unary -, ~	6
* /	5
+ -	4
<, \leq , =, \neq , \geq , >	3
AND	2
OR	1

 Here you'll get 

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

@ENGINEERINGWALLAH
By Ganesh Bodakhe



Algorithm Infix to postfix conversion

Step 1 : The input string (infix notation) is scanned from left to right.

Step 2 : If the scanned character (ch) is space or tab, it is skipped.

Step 3 : If the scanned character (ch) is digit or alphabet, it is appended to postfix string.

Step 4 : If the scanned character (ch) is opening parenthesis '(', it is pushed to stack.

Step 5 : If the scanned character (ch) is operator :

All operators from top of the stack are popped having more or same priority than ch and appended to postfix string.

When less priority operator is found, then it is pushed in the stack again with ch.

Step 6 : If the scanned character (ch) is closing parenthesis ')', then all the operators above the opening parenthesis are appended to postfix string.

Step 7 : After evaluations of all the characters, the remaining operators from stack are appended to postfix string.

Infix to postfix conversion

- Manual algorithm for converting infix to postfix

(a + b)* c

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

DIGITAL INSTITUTE
By Ganesh Bodashe

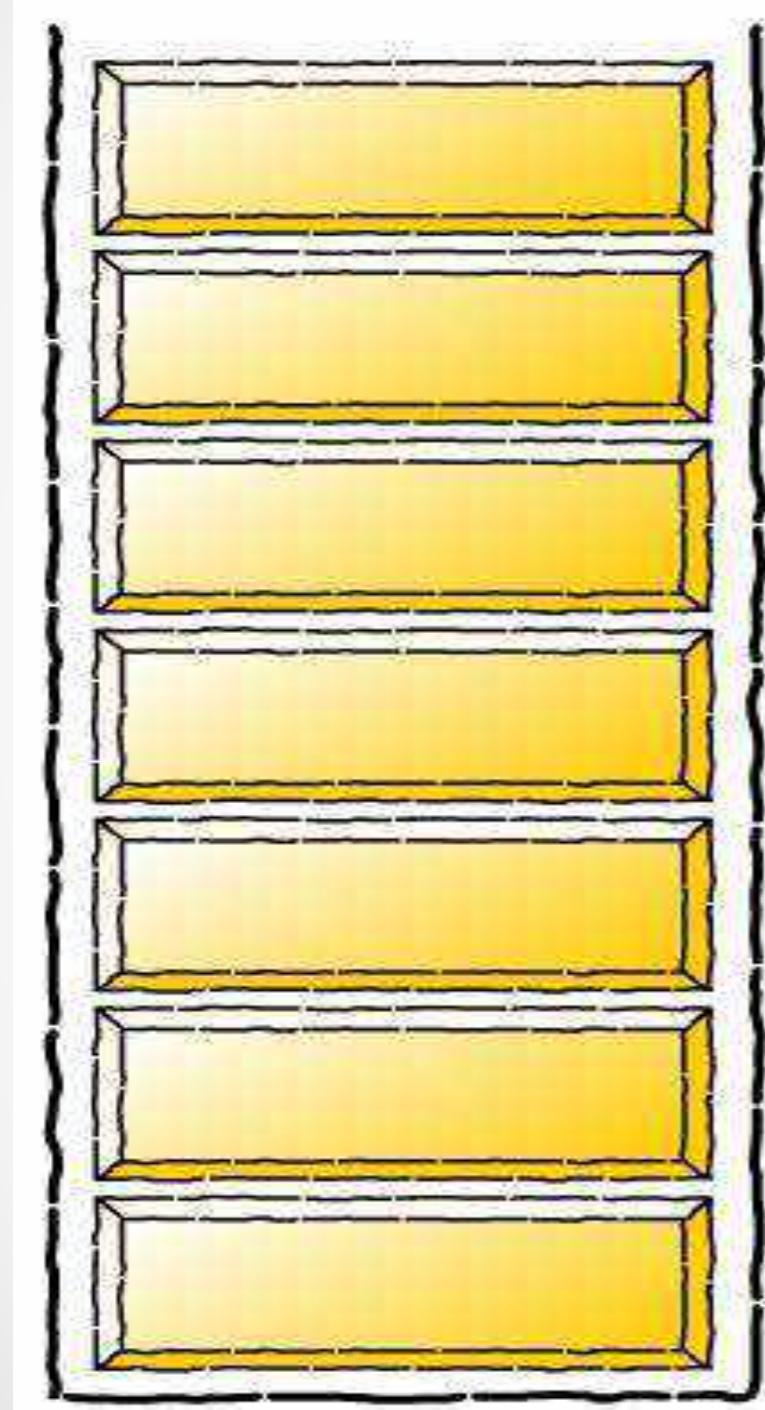
- Write with parentheses to force correct operator precedence $((a+b)^* c)$

- Move operator to right inside parentheses
 $((a b +) c ^*)$

- Remove parentheses

a b + c *

Infix to postfix conversion



infixVect

(E_W a + b - c) * d - (e + f)

postfixVect

Here you'll get

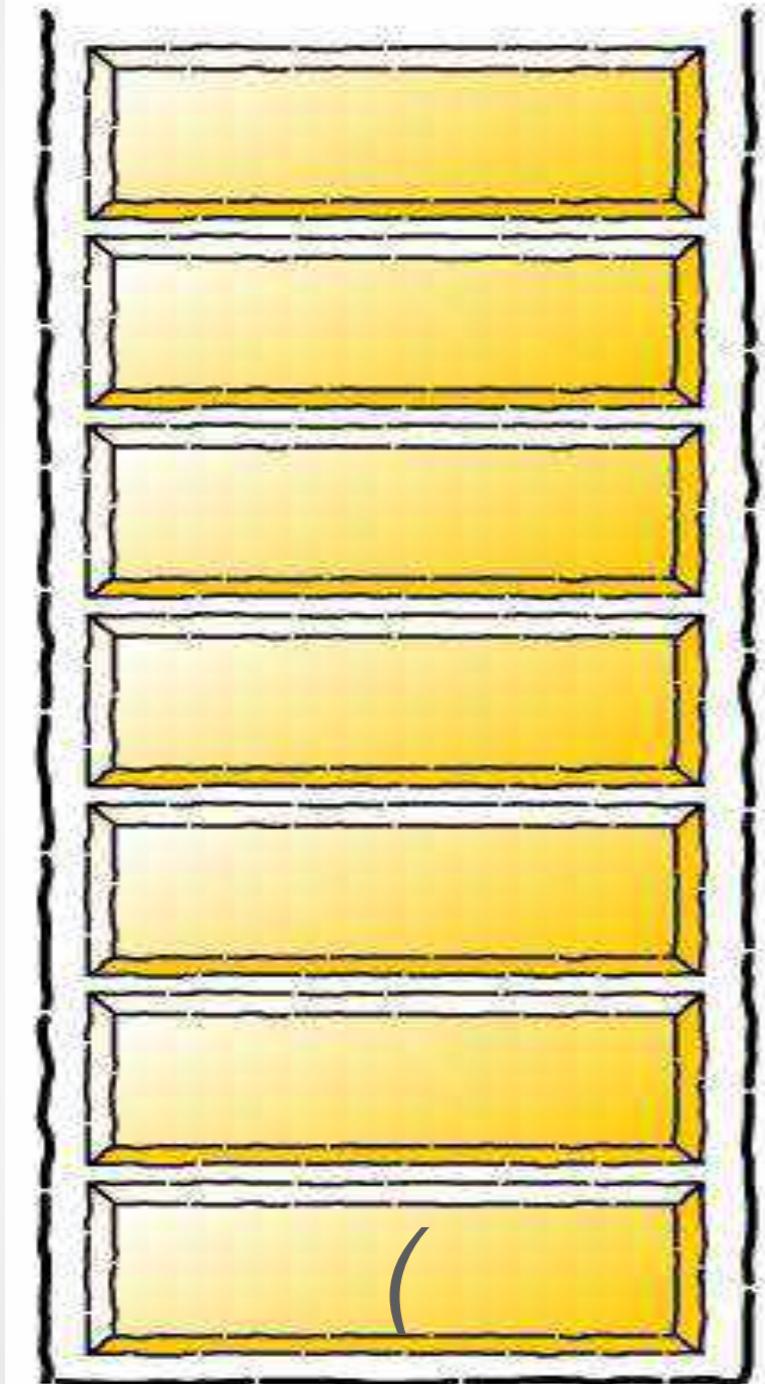
- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

$a + b - c) * d - (e + f)$

postfixVect

Here you'll get

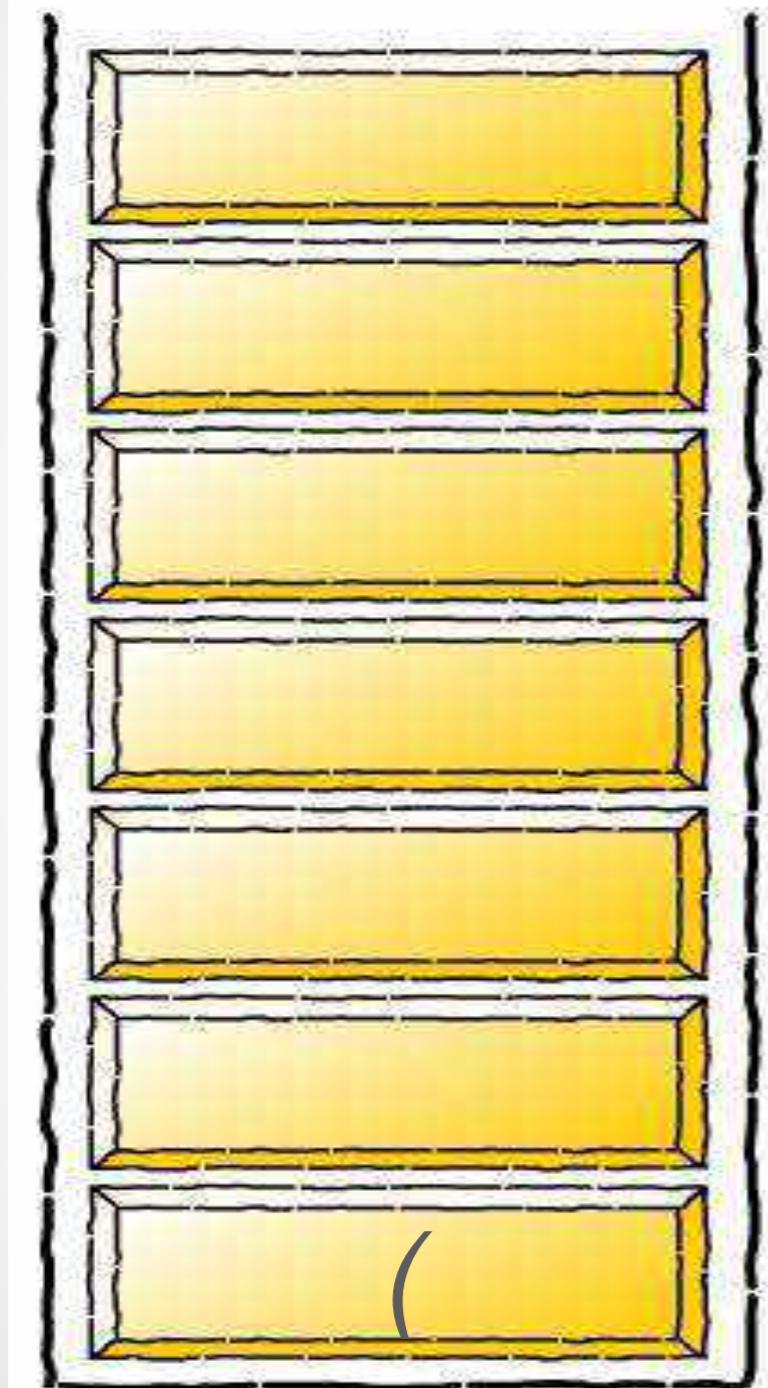
- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

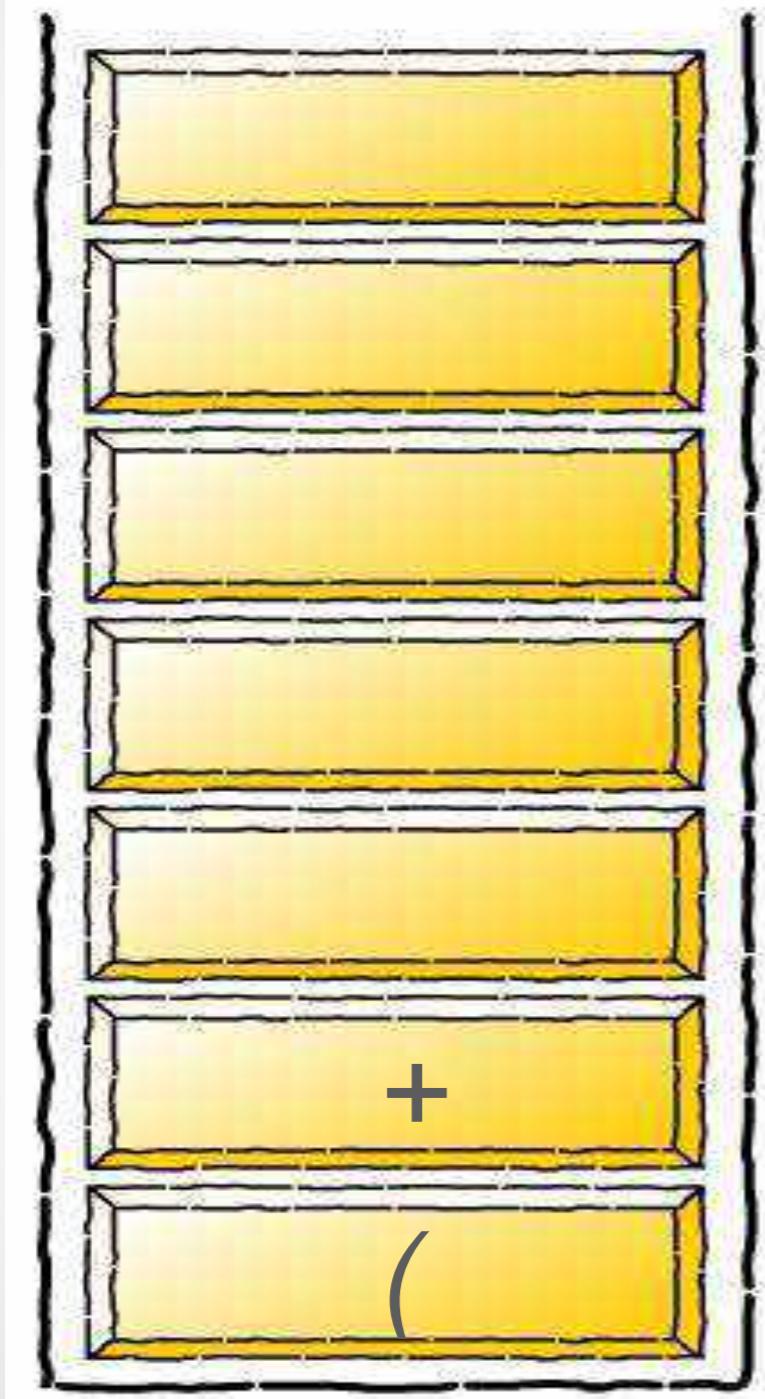
+EW b - c) * d - (e + f)

Here you'll get
• PPT
• NOTES
• VIDEO LECTURE
• E-BOOK
• PYQ
• EXPERIMENT
• ASSIGNMENT
• TUTORIAL

EW
a
@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

bEWc) * d - (e + f)

postfixVect

Here you'll get

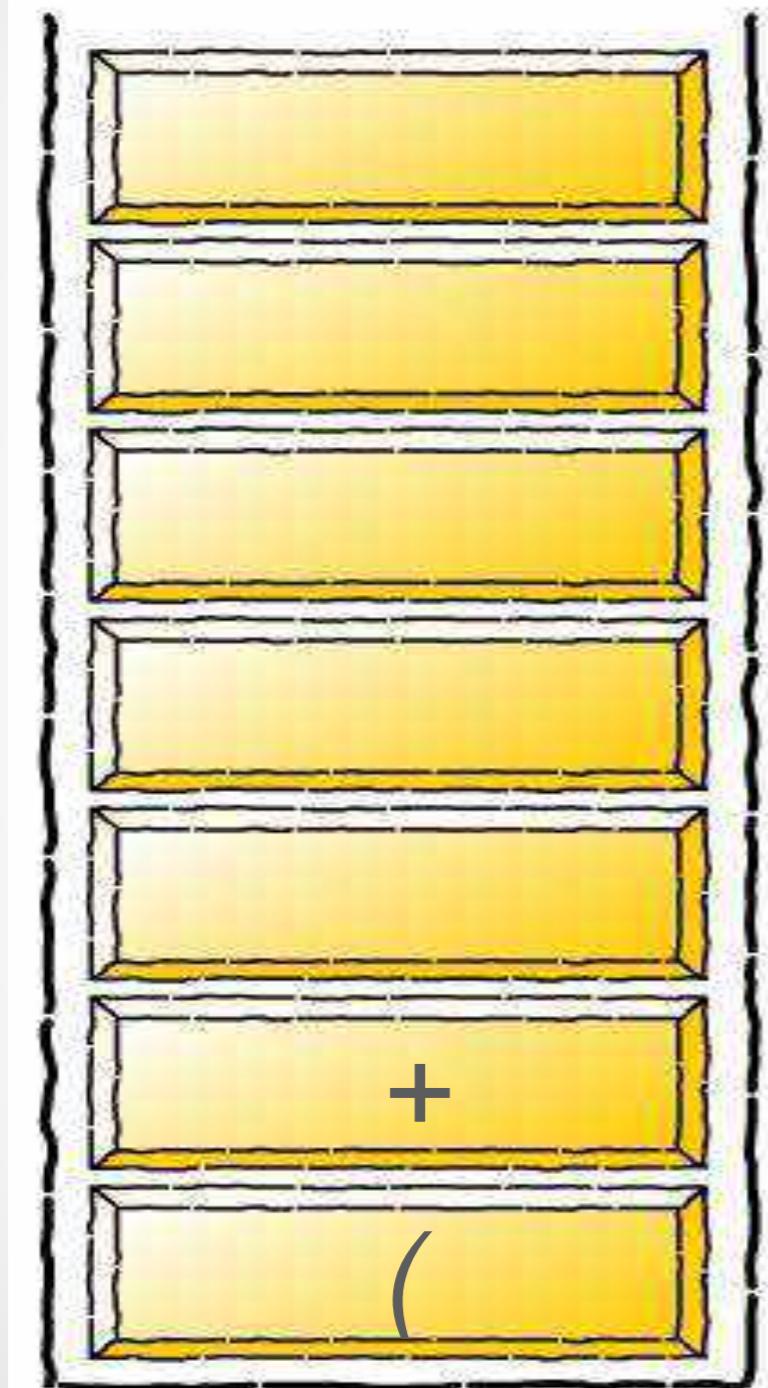
- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

- c) * d - (e + f)

Here you'll get
EngineeringWallah

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

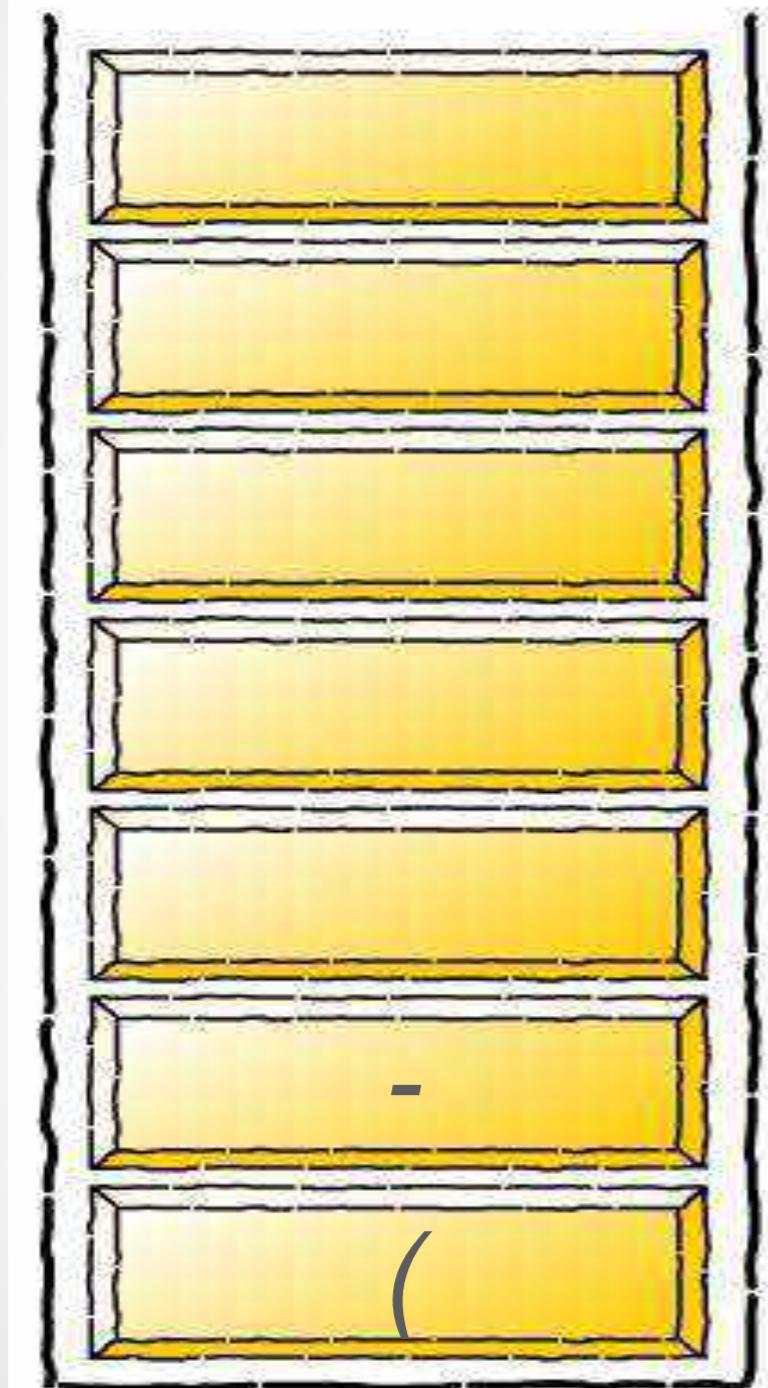
a b



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

cEW* d - (e + f)

Here you'll get
EngineeringWallah

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

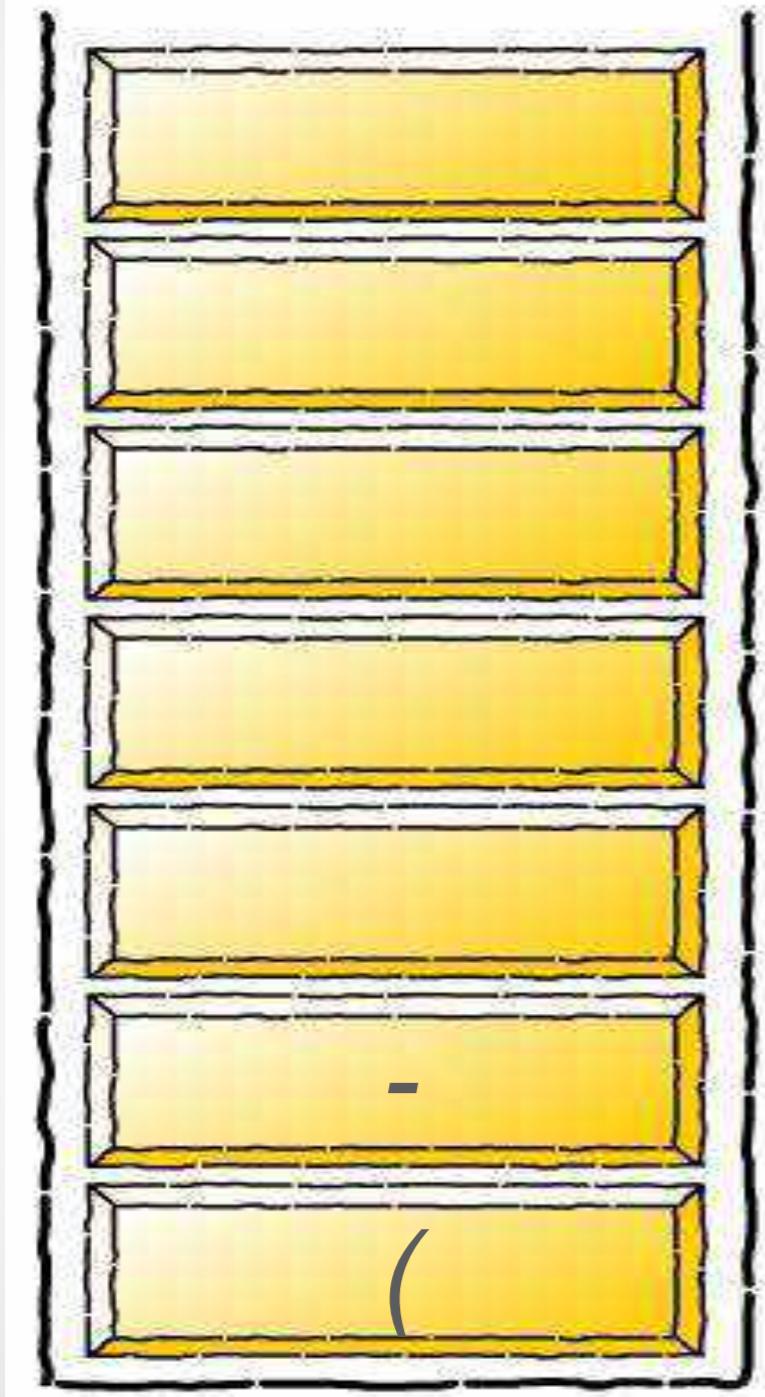
a b +



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

)E*Wd - (e + f)

postfixVect

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

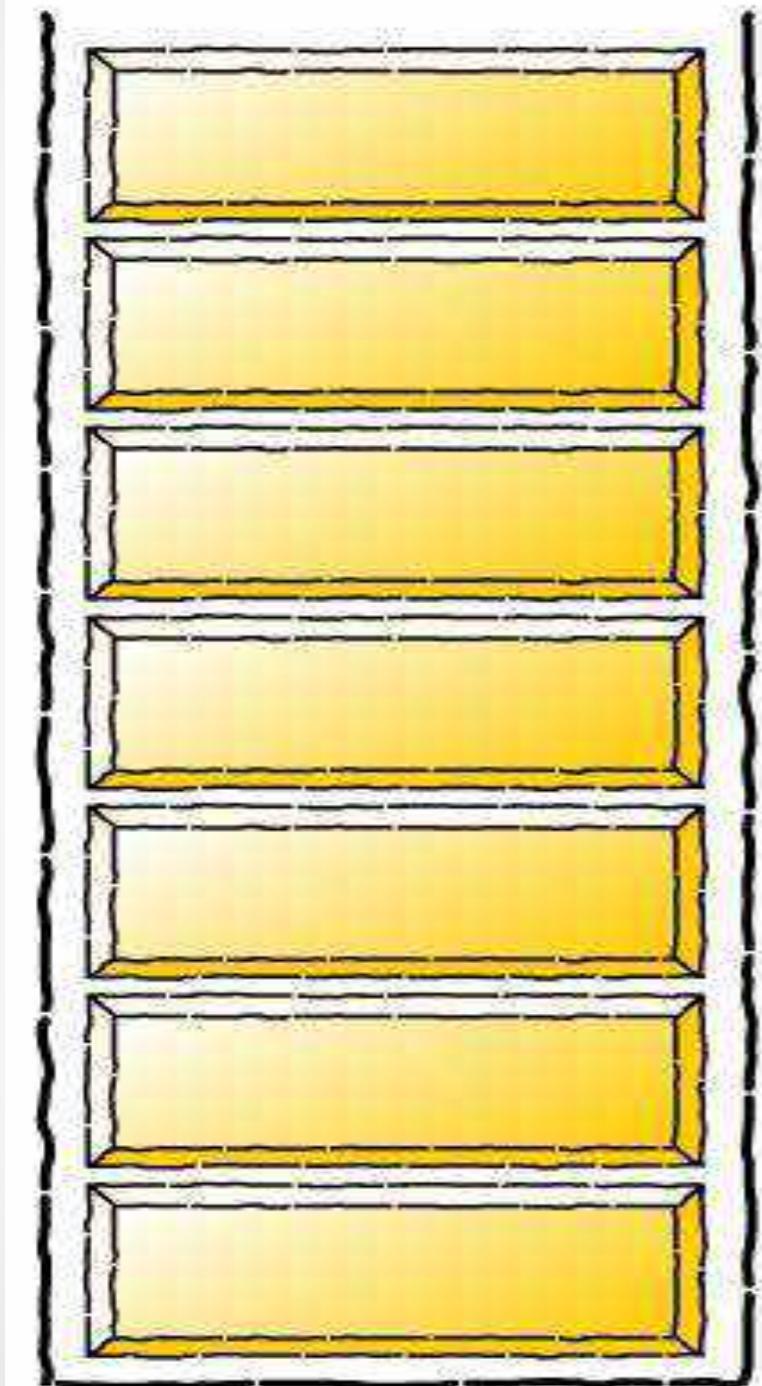
a b + c



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect



(*EdW) (e + f)

postfixVect

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

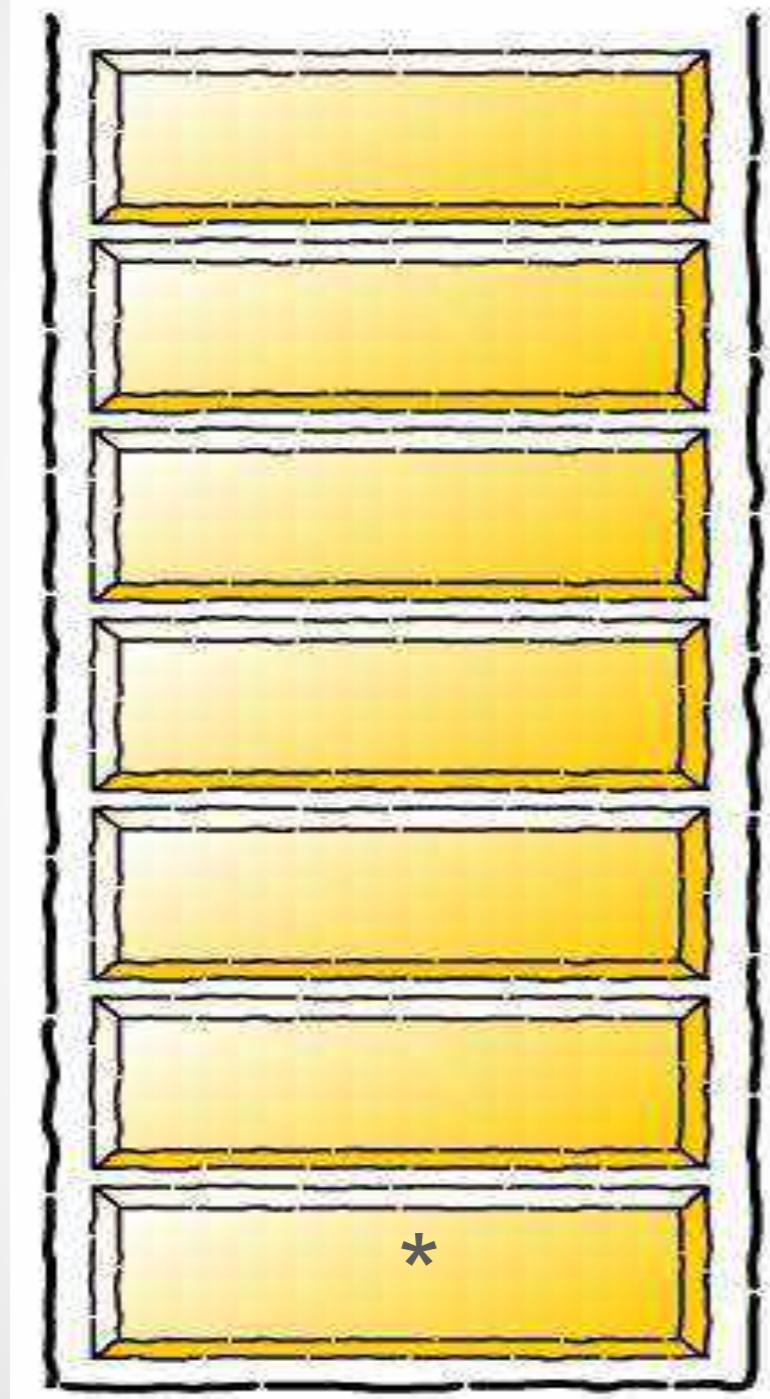
a b + c



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

dEW(e + f)

postfixVect

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

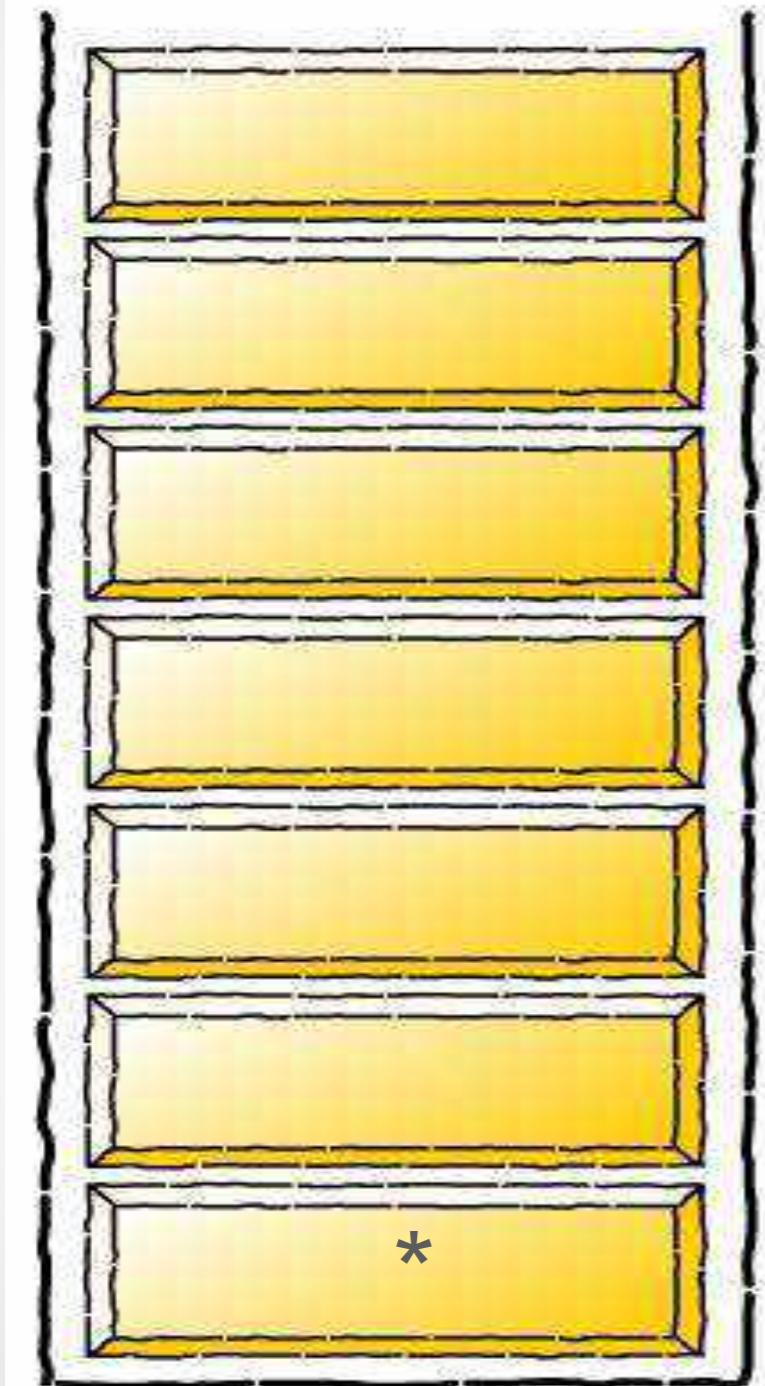
a b + c



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

-EW(e + f)

postfixVect

- Here you'll get
- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

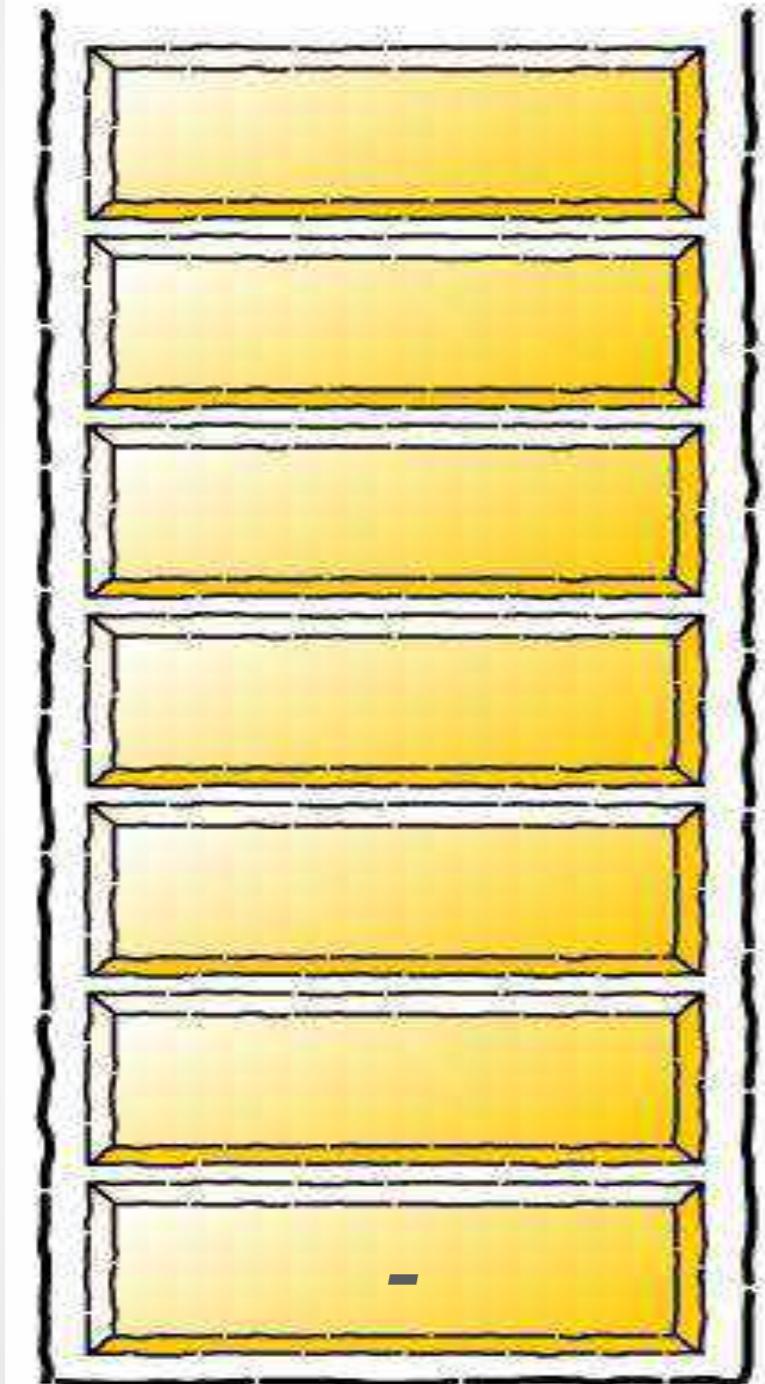
a b + c - d

@ENGINEERINGWALLAH

By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

(e + f)

postfixVect

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

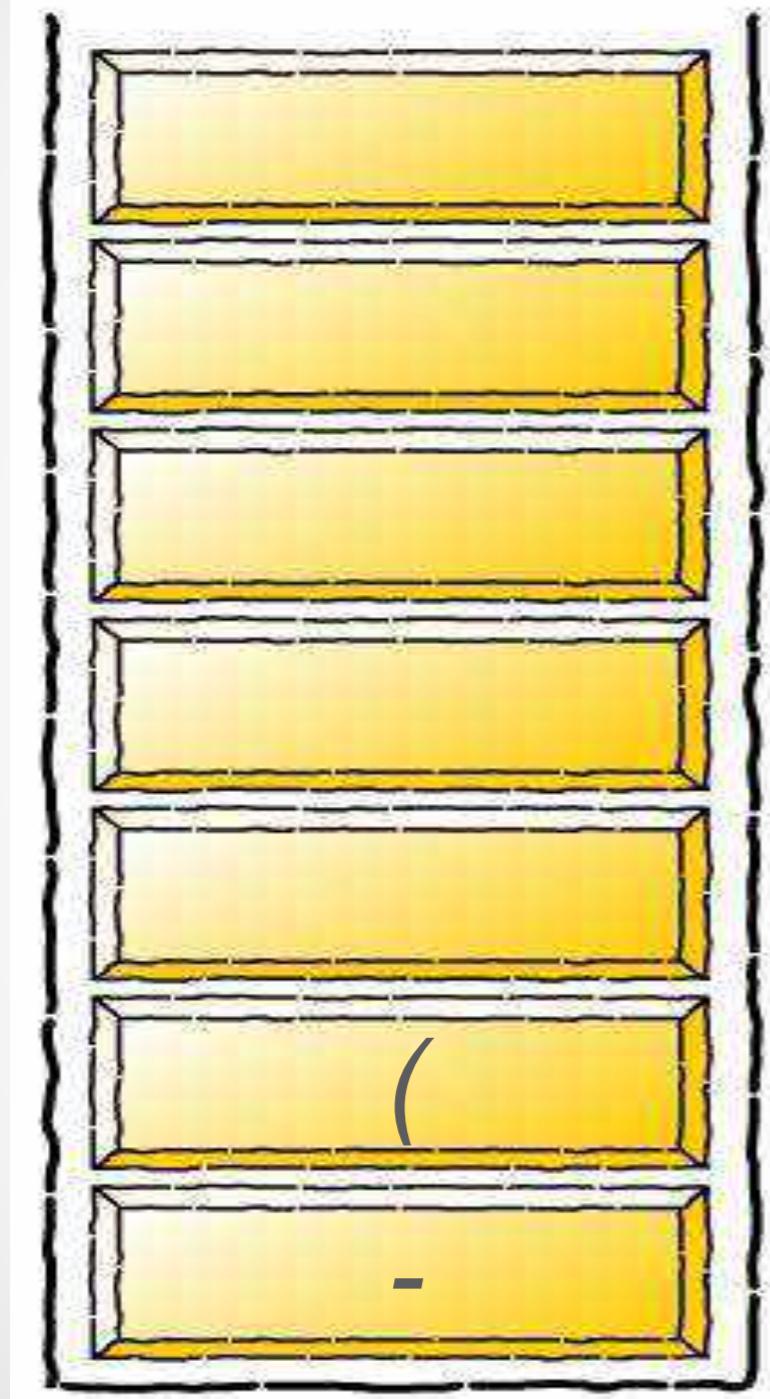
a b + c d *



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

e + f)

postfixVect

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

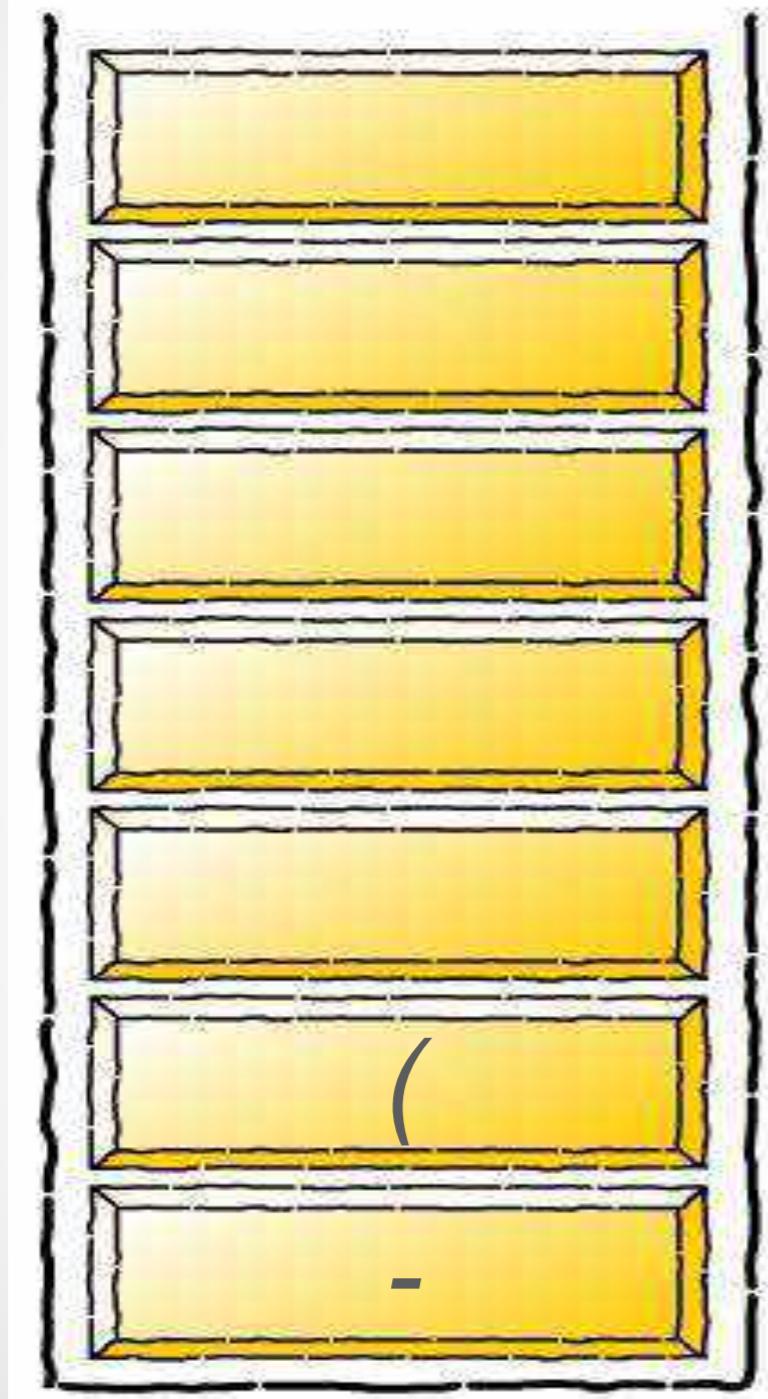
a b + c d *



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

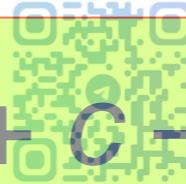


postfixVect

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

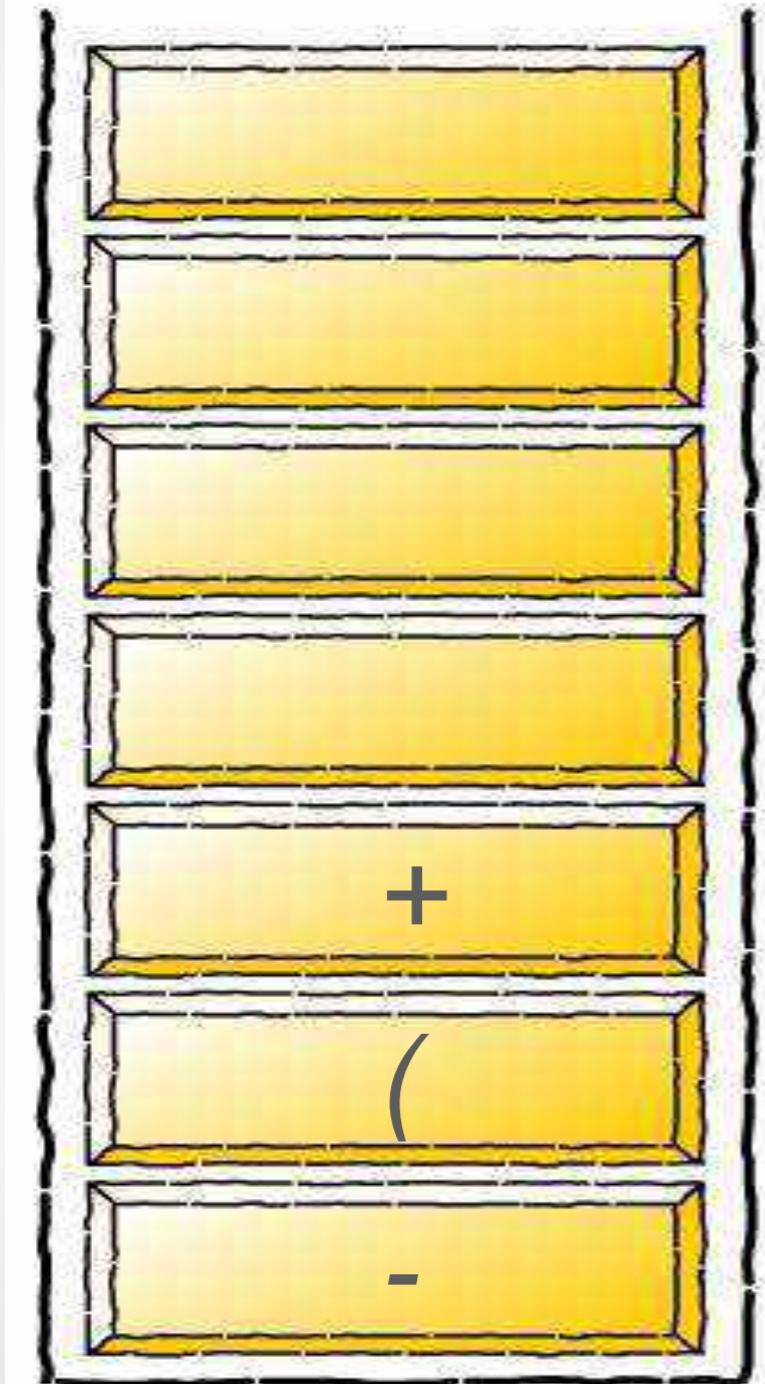
a b + c d * e



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect



postfixVect

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

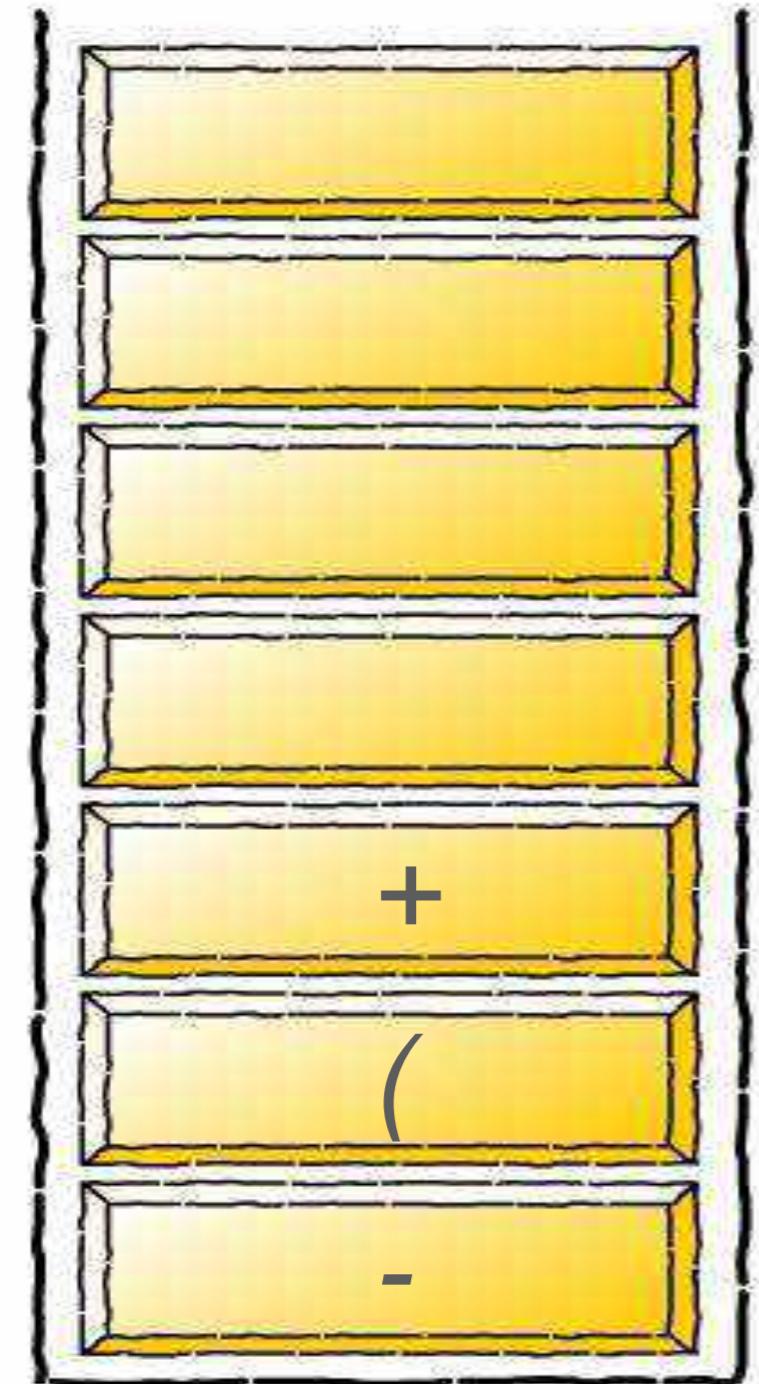
a b + c d * e



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect



postfixVect

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

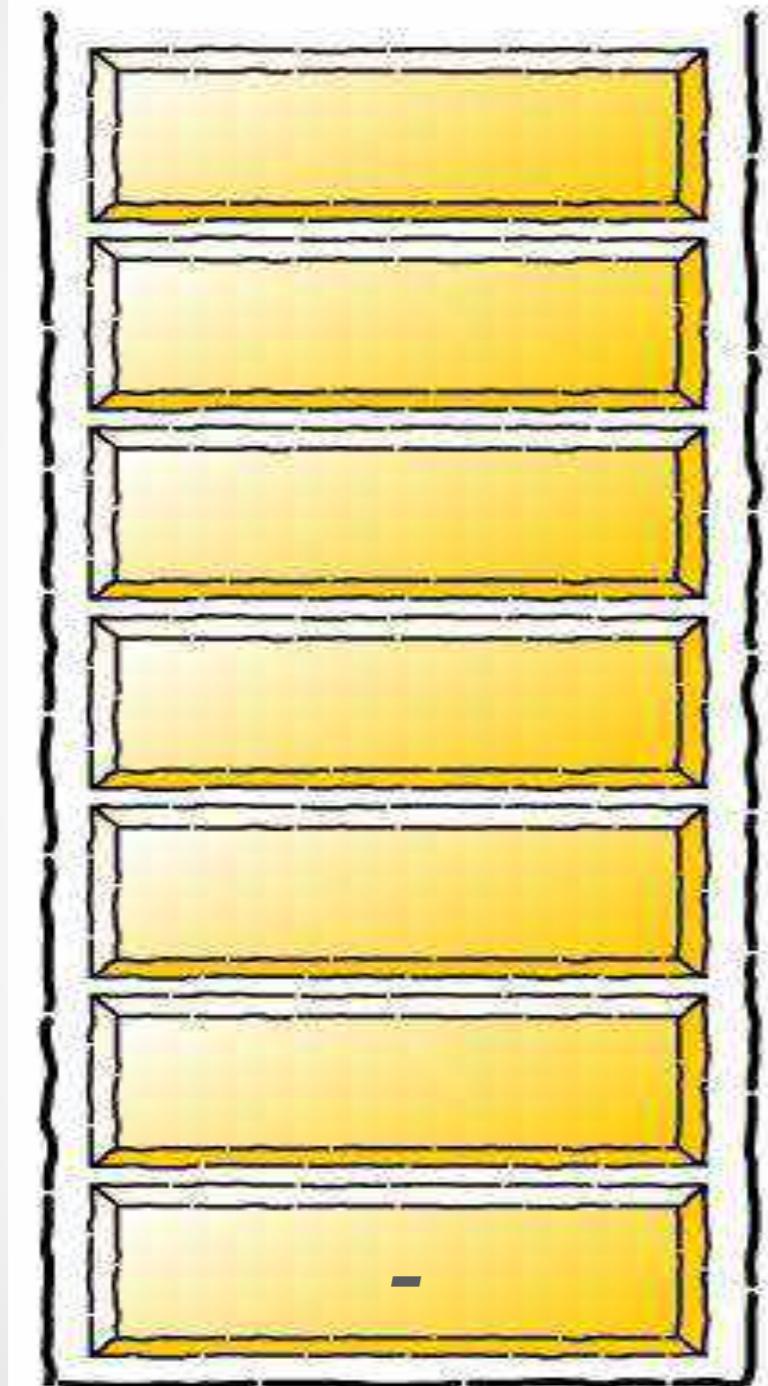
a b + c d * e f



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

EW

postfixVect

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

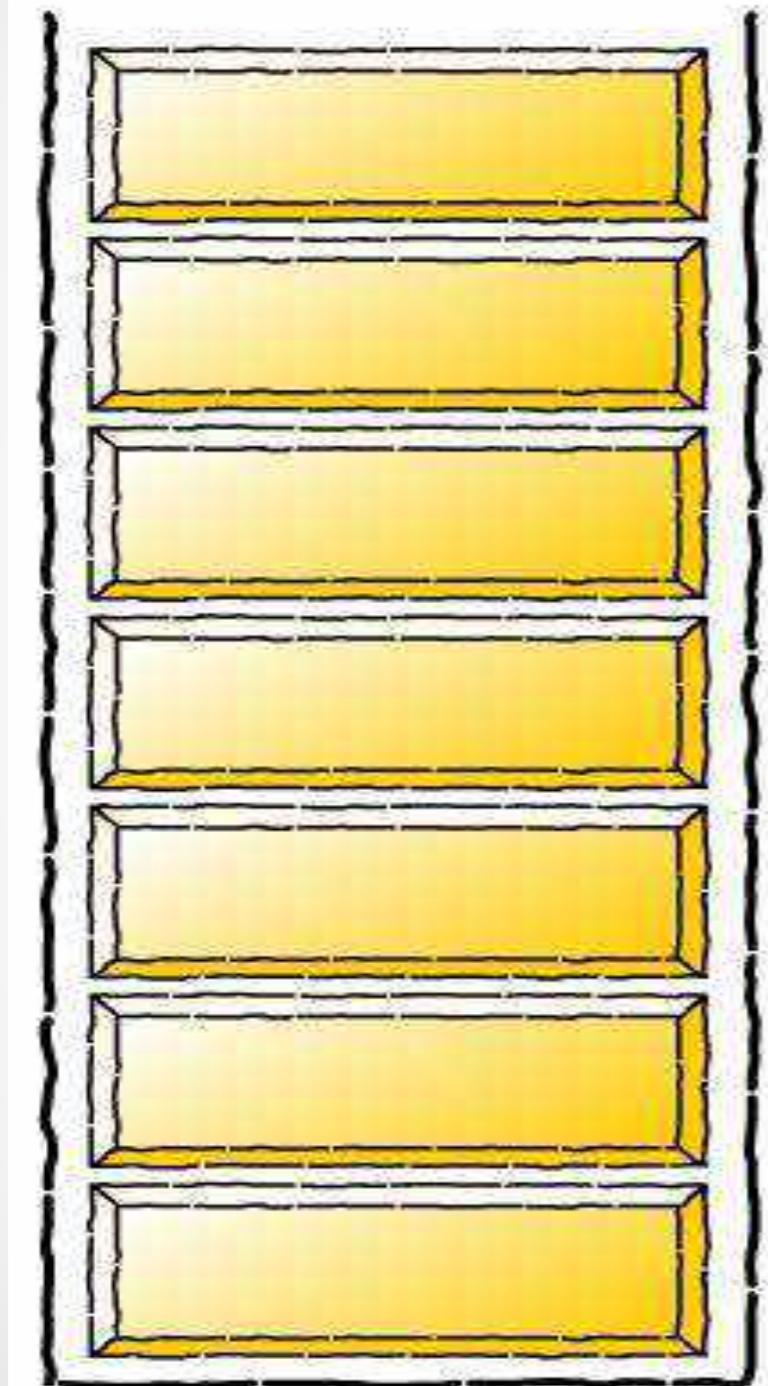
a b + c - d * e f +



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

stackVect



infixVect

EW

Here you'll get
EW
postfixVect

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

a b + c - d * e f + -



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Infix to postfix conversion

Infix Expression: $a + (b^*c)$.

Contin....

Conversion of Infix to Postfix

Example to Convert Infix to Postfix using stack

a + (b*c)

Read character	Stack	Output
a	Empty	a
+	+ (a
(+ (a
b	+ (b	ab
*	+ (b *	ab
c	+ (b * c	abc
)	+ (b * c)	abc*
		abc*+

Infix to postfix conversion

Infix Expression: A * B + C * D.

Character Scanned	Stack	Postfix)
A	-	A
*	EW*	A
B	*	AB
+	+ EW	AB*
C	+ +	AB*C
*	+*	AB*C
D	+*	AB*CD
	Empty	AB*CD*+

Postfix expression is = AB*CD*+

• Result

Infix to postfix conversion

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

Table 4.8.1 : Infix to Postfix Conversion

Character from Infix	Stack	Postfix Expression
((
(((
A	((A	
+	((+ A	
B	((+ AB	
)	(AB+	
*	(* AB+	

Character from Infix	Stack	Postfix Expression
((*()	AB+
C	(*()	AB+C
-	(*(-	AB+C
D	(*(-	AB+CD
)	(*	AB+CD-
/	/	AB+CD-*
E	/	AB+CD-*E
		AB+CD-*E/

- Resultant Postfix Expression: $AB+CD-*E/$

Infix to postfix conversion

$a \uparrow b * c - d + e / f / (g + h)$

Table 4.8.2 : Infix to Postfix Conversion

Character Scanned	Stack	Postfix)
A	-	a
\uparrow	\uparrow	a
B	\uparrow	ab
*	\uparrow	ab
C	*	ab \uparrow c
-	-	ab \uparrow c*
D	-	ab \uparrow c * d
+	+	ab \uparrow c * d -
E	+	ab \uparrow c * d - e
/	+/-	ab \uparrow c * d - e
F	+/-	ab \uparrow c * d - ef
/	+/-	ab \uparrow c * d - ef /
(+/(ab \uparrow c * d - ef /
G	+/(ab \uparrow c * d - ef / g

Character Scanned	Stack	Postfix)
+	+/(+	ab \uparrow c * d - ef / g
H	+/(+	ab \uparrow c * d - ef / gh
	+ /	ab \uparrow c * d - ef / gh +
	Empty	ab \uparrow c * d - ef / gh + / +

∴ Postfix expression is = ab \uparrow c * d - ef / gh + / +

• Resultant Postfix Expression:

Infix to postfix conversion

$((A + B) * D) \uparrow (E - F)$

Table 4.8.3 : Infix to Postfix Conversion

Character Scanned	Stack	Postfix
((-
(((-
A	((A
+	((+	A
B	((+	AB
)	(AB+
*	(*	AB+
D	(*D	AB+D
)	Empty	AB+D*
\uparrow	\uparrow	AB+D*
(\uparrow (AB+D*E
E	\uparrow (AB+D*E
-	\uparrow (-	AB+D*E
F	\uparrow (-	AB+D*EF
	\uparrow	AB+D*EF-
	Empty	AB+D*EF- \uparrow

\therefore Postfix expression = AB+D*EF- \uparrow

Infix to postfix conversion

$((a/(b-c+d))^{*}(e - a)^{*}c)$

Character Scanned	Stack	Postfix
((-
(((-
A		a
/	((/	a
(((/()	a
B		ab
-	((/(-	
C		abc
+	((/(+	abc-
D	((/(+	abc-d

)	((abc-d+
)		abc-d+/-
*	(*	abc-d+/-
((*(abc-d+/-
E	(*(abc-d+/-e
-	(*(-	abc-d+/-e
A	(*(-	abc-d+/-ea
)	(*	abc-d+/-ea-
*	(*	abc-d+/-ea-*
c	(*	abc-d+/-ea-*c
)	empty	abc-d+/-ea-*c*

- Resultant Postfix Expression: **$abc-d+/-ea-*c^*$**

Infix to postfix conversion

Infix Expression: $A + (B^*C - (D/E^F)^*G)^*H$, where \wedge is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(Start
2.	A	(A	
3.	+	(+	A	
4.	((+(A	
5.	B	(+(AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	** is at higher precedence than -
9.	((+(-()	ABC*	
10.	D	(+(-()	ABC**D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	\wedge	(+(-(/ \wedge	ABC*DE	
14.	F	(+(-(/ \wedge	ABC*DEF	
15.)	(+(-	ABC*DEF \wedge /	Pop from top on Stack, that's why ' \wedge ' Come first
16.	*	(+(-*	ABC*DEF \wedge /	
17.	G	(+(-*	ABC*DEF \wedge /G	
18.)	(+	ABC*DEF \wedge /G*-	Pop from top on Stack, that's why ' \wedge ' Come first
19.	*	(+*	ABC*DEF \wedge /G*-	
20.	H	(+*	ABC*DEF \wedge /G*-H	
21.)	Empty	ABC*DEF \wedge /G*-H*+	END

Resultant Postfix Expression: $ABC*DEF\wedge/G^*-H^*+$

Infix to postfix conversion

Infix Expression: $A + (B^*(C-D)/E)$.

	RPN	Stack	Input Expression	RPN	Stack	Input Expression
1		($A + (B^*(C-D)/E)$			$D) / E)$
2	A	(A	$+ (B^*(C-D)/E)$			$) / E)$
3	A	(+ A	$(B^*(C-D)/E)$			$/ E)$
4	A	(+ B	$B^*(C-D)/E)$			$)$
5	AB	(+ B	$*(C-D)/E)$			
6	AB	(* B	$(C-D)/E)$			
7	AB	(* C	$(C-D)/E)$			
8	ABC	(* C	$-D)/E)$			
9	ABC					
10	ABCD					
11	ABCD-					
12	ABCD-*					
13	ABCD-*E					
14	ABCD-*E/					
15	ABCD-*E/+					



- Here you'll get:
- PPT
 - NOTES
 - VIDEO LECTURE
 - E-BOOK
 - PYQ
 - EXPERIMENT
 - ASSIGNMENT
 - TUTORIAL



@ENGINEERINGWALLAH
By Ganesh Bodakhe

Resultant Postfix Expression: $ABCD-*E/+$

Infix to postfix conversion

Infix Expression: A * (B + C * D) + E.

Infix to Postfix using stack ...



- Example A * (B + C * D) + E becomes A B C D * + * E +

	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	(*	A
4	B	*	A B
5	+	*	A B
6	C	*	A B C
7	*	*	A B C
8	D	*	A B C D
9)		A B C D * +
10	+	+	A B C D * + *
11	E	+	A B C D * + * E
12			A B C D * + * E +

- Resultant Postfix Expression: ABCD*+*E+

Infix to postfix conversion

Infix Expression : $A+B*(C^D-E)$

Token	Action	Result	Stack	Notes
A	Add A to the result	A		
+	Push + to stack	A	+	
B	Add B to the result	AB	+	
*	Push * to stack	EW AB	* +	* has higher precedence than +
(Push (to stack	AB	(* +	
C	Add C to the result	ABC	(* +	
^	Push ^ to stack	ABC	EW ^ (* +	
D	Add D to the result	ABCD	^ (* +	
-	Pop ^ from stack and add to result. Push - to stack	ABCD ^A	(* +	- has lower precedence than ^
E	Add E to the result	ABCD ^A E	- (* +	
)	Pop - from stack and add to result Pop (from stack	ABCD ^A E-	(* +	Do process until (is popped from stack
	Pop * from stack and add to result	ABCD ^A E-*	+ Given expression is iterated, do	
	Pop + from stack and add to result	ABCD ^A E-*+		Process till stack is not Empty, It will give the final result

Postfix Expression : ABCD^AE-*+

Infix to postfix conversion

Suppose we want to convert $2 * 3 / (2 - 1) + 5 * 3$ into Postfix form,

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
((23*
2	(23*2
-	(-	23*2
1	(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

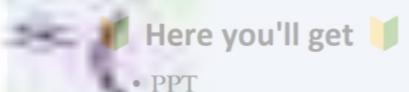
So, the Postfix Expression is $23 * 21 / 53 - 53 +$

Infix to postfix conversion

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

to postfix notation.

Scanned character	stack	postfix
A	EW	A
*		A
(EW	A
B	EW	AB
+	EW	AB
C	EW	ABC
)		ABC+
*		ABC+*
D		ABC+*D
	Empty	ABC+*D*



Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



@ENGINEERINGWALLAH
By Ganesh Bodakhe

POSTFIX EXPRESSION EVALUATION

ALGORITHMS :

- Step 1** : Read postfix expression from Left to Right
- Step 2** : If operand is encountered, push it in Stack.
- Step 3** : If operator is encountered, Pop two elements
 $A \rightarrow$ Top element $B \rightarrow$ Next Top element
Evaluate B operator A
- Step 4** : Push result into stack
- Step 5** : Read next postfix element if not end of postfix string
- Step 6** : Repeat from Step 2
- Step 7** : Print the result popped from stack.

Here you'll get

- NOTE
- VIDEO LECTURE
- E-BOOK
- PDF
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

@ENGINEERINGWALLAH
By Ganesh Bodakhe



POSTFIX EXPRESSION EVALUATION

POSTFIX EVALUATE EXPRESSION : $53+82-*$

Reading symbol	Stack operations	Evaluated part of expression			
Initially	Stack is Empty	Nothing	8	push(8)	$(5 + 3)$
5	push(5)	Nothing	2	push(2)	$(5 + 3)$
3	push(3)	Nothing		Value 1 = pop(); // 2 Value 2 = pop(); // 8 result = 8 - 2; // 6 Push(6) $(8 - 2)$ $(5 + 3), (8 - 2)$	Value 1 = pop(); // 2 Value 2 = pop(); // 8 result = 8 - 2; // 6 Push(6) $(8 - 2)$ $(5 + 3), (8 - 2)$
+	Value 1 = pop(); Value 2 = pop(); result = Value 2 + Value 1 push(result)	Value 1 = pop(); // 3 Value 2 = pop(); // 5 result = 5 + 3; // 8 Push(8) $(5 + 3)$		Value 1 = pop(); Value 2 = pop(); // 8 result = Value 2 + Value 1 push(result)	Value 1 = pop(); // 6 Value 2 = pop(); // 8 result = 8 * 6; // 48 Push(48) $(6 * 8)$ $(5 + 3), (8 - 2)$
		End of expression		result = pop()	Display (result) 48 as final result

POSTFIX EXPRESSION EVALUATION

* Evaluate the following postfix expression and show stack after every step in tabular form. Given A=5, B=6, C=2, D=12, E=4

ABC + *DE\ -

ABC+*DE\/-

This will become

5 6 2 + * 12 4 / -

Table 4.9.2 : Evaluate Postfix Expression

Reading	Stack	Evaluated
5 • PPT • NOTES • VIDEO LECTURE • E-BOOK • PYQ • EXPERIMENT • ASSIGNMENT • TUTORIAL	5	
6	5,6	
2	5,6,2	
+	5,8	$6 + 2 = 8$
*	40	$5 * 8 = 40$
12	40,12	
4	40,12,4	
/	40,3	$12 / 4 = 3$
-	37	$40 - 3 = 37$

Result is 37.

POSTFIX EXPRESSION EVALUATION

- Evaluate the following postfix expression

A : 6, 2, 3, +, - , 3, 8, 2, +, +, *, 2, ^, 3, +

Reading	Stack	Evaluated
6	6	
2	6 2	
3	6 2 3	
+	6 5	$2 + 3 = 5$
-	1	$6 - 5$
3	1 3	
8	1 3 8	
2	1 3 8 2	
+	1 3 10	$8 + 2 = 10$
+	1 13	$3 + 10 = 13$
*	13	$1 * 13$
2	13 2	

(Sem 3-Comp.)

Reading	Stack	Evaluated
169		$13^2 = 169$
3	169 3	
+	172	$169 + 3 = 172$

Result 172

@ENGINEERINGWALLAH
By Ganesh Bodakhe

POSTFIX EXPRESSION EVALUATION

- Consider the following arithmetic expression written in postfix notation
 $10, 2, *, 15, 3, /, +, 12, 3, 2, \uparrow, +, +$ evaluate this expression to find its value

Reading	Stack	Evaluated
10	10	
2	10 2	
*	20	$10 * 2 = 20$
15	20 15	
3	20 15 3	
/	20 5	$15 / 3 = 5$
+	25	$20 + 5$
12	25 12	
3	25 12 3	
2	25 12 3 2	
\uparrow	25 12 9	$3^2 = 9$
+	25 21	$12 + 9 = 21$
+	46	$25 + 21 = 46$

Here you'll get

• PPT

• NOTES

• VIDEO LECTURE

• E-BOOK

• PYQ

• EXPERIMENT

• ASSIGNMENT

• TUTORIAL

20 15 3

20 5

25

25 12

25 12 3

25 12 3 2

25 12 9

25 21

46

EW

QR

Code

By Ganesh Bodakhe

@ENGINEERINGWALLAH

Convert infix into prefix expression

convert the given infix to prefix expression and show detail of stack.

(A-B/C)*(D*E-F)

First we have to reverse the string

(F - E * D) * (C/B - A)

Table 4.10.1 : Evaluation of Prefix Expression

Character scanned	Stack	Postfix
((-
F	(F
-	(-	F
E	(-	FE
*	(-*	FE
D	Here you'll get • PPT • NOTES • VIDEO LECTURE • E-BOOK	FED
)	Empty	FED*- -
*	*	FED*- -
(* (FED * -
C	* (FED * - C
/	* (/	FED * - C
B	* (/	FED * - CB
-	* (-	FED * - CB/
A)	* (-	FED * - CB/A
)	*	FED * - CB/A -
End	Empty	FED * - CB/A - *

Now reverse the expression :

Prefix expression : * - A/BC - * DEF

Convert infix into prefix expression

convert infix string $((A+B) * (C-D))/(E+F)$ into prefix string with stack.

→ first reverse the string $(F+E)/((D-C) * (B+A))$

Character Scanned	Stack	Postfix
((-
F	(F	F
+	(+ F	F
E	(+(E	FE
)	Empty	FE+
/	/	FE+
(/(FE+
(/((FE+
D	/((D	FE+D
-	/((-	FE+D
C	/((- C	FE+DC
)	/((-)	FE+DC-
*	/((-) *	FE+DC-
(/((-) (FE+DC-
B	/((-) (B	FE+DC-B
+	/((-) (+	FE+DC-B
A	/((-) (+ A	FE+DC-BA
)	/((-) *	FE+DC-BA+
)	/	FE+DC-BA+*
	Empty	FE+DC-BA+*/

Reverse the string

Result will be : /* + AB - CD + EF

Convert prefix into postfix expr.

ALGORITHMS :

- Read the Prefix expression in reverse order (from right to left)
- If the symbol is an operand, then push it onto the Stack
- If the symbol is an operator, then pop two operands from the StackCreate a string by concatenating the two operands and the operator after them.string = operand1 + operator + operand2 And push the resultant string back to Stack.
- Repeat the above steps until end of Prefix expression.

Convert prefix into postfix expr.

Convert the following prefix expression into postfix expression

$*+a-bc/-de+-fgh$

Expression scanned from right to left.

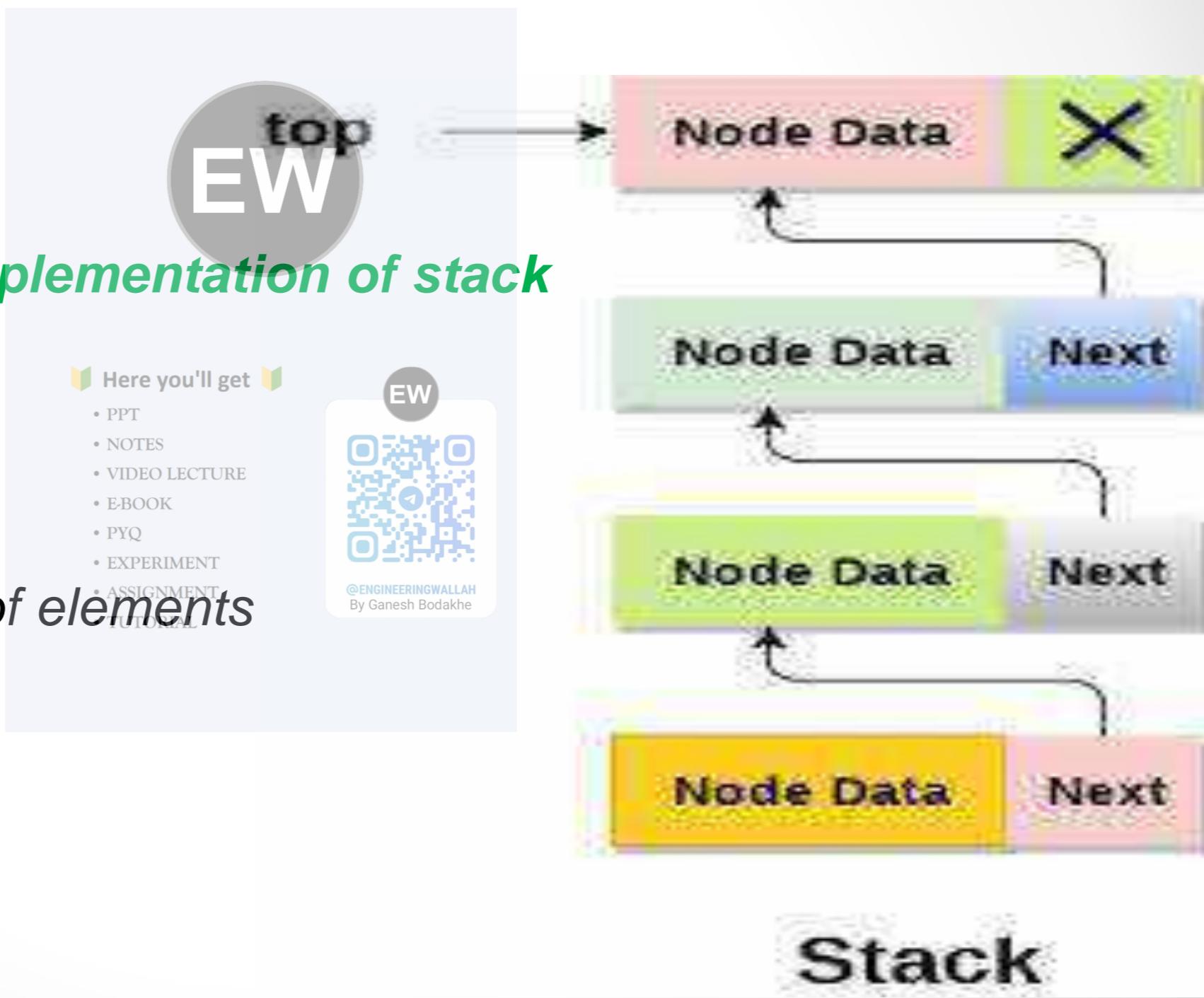
Prefix	Stack	Postfix
$*+a-bc/-de+-fgh$	-	-
$*+a-bc/-de+-fg$	h	-
$*+a-bc/-de+-f$	gh	-
$*+a-bc/-de+-$	fg	-
$*+a-bc/-de+$	fg-h	fg-
$*+a-bc/-de$	fg-h+	fg-h+
$*+a-bc/-d$	e fg-h+	fg-h+
$*+a-bc/-$	d e fg-h+	fg-h+
$*+a-bc/$	de- fg-h+	fg-h+

Prefix	Stack	Postfix
$*+a-bc$	de-fg-h+/	de-fg-h+/
$*+a-b$	c de-fg-h+/	de-fg-h+/
$*+a-$	b c de-fg-h+/	de-fg-h+/
$*+a$	bc- de-fg-h+/	de-fg-h+/
$*$	a bc- de-fg-h+/	de-fg-h+/
Empty	Empty	abc+ de-fg-h+/*

∴ Postfix expression = AB+D*EF-↑

Linked stack operation

In stack elements are placed one above other. In the same manner in stack as linked list, we place node one above other.



Advantages of dynamic implementation of stack

- 1. No memory wastage
- 2. No memory shortage
- 3. No limitation on number of elements



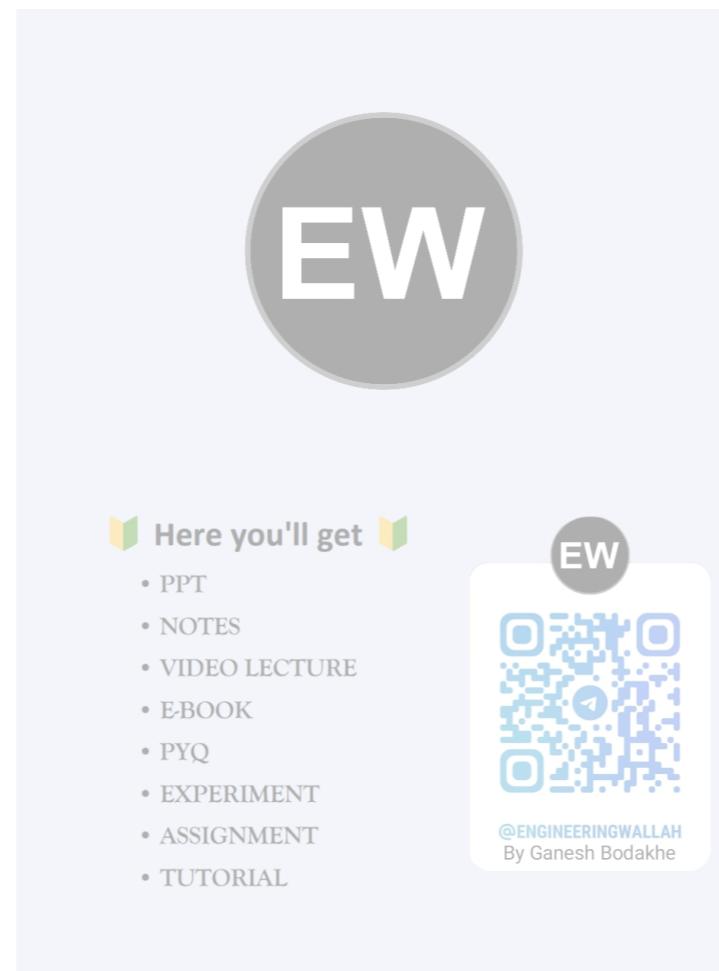
Stack operation using linked list

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    struct Node *next;
};

struct Node* top = NULL;

void push(int val) {
    struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
    newnode->data = val;
    newnode->next = top;
    top = newnode;
}
```



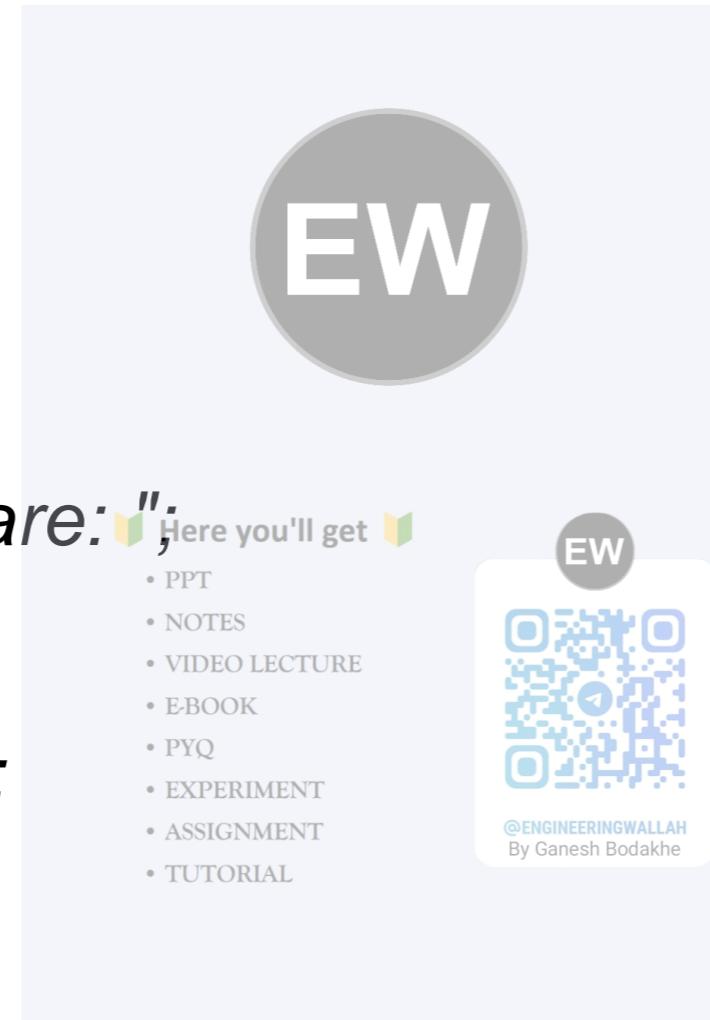
Stack operation using linked list

```
void pop() {  
    if(top==NULL)  
        cout<<"Stack Underflow"<<endl;  
    else {  
        cout<<"The popped element is "<< top->data << endl;  
        top = top->next;  
    }  
}
```



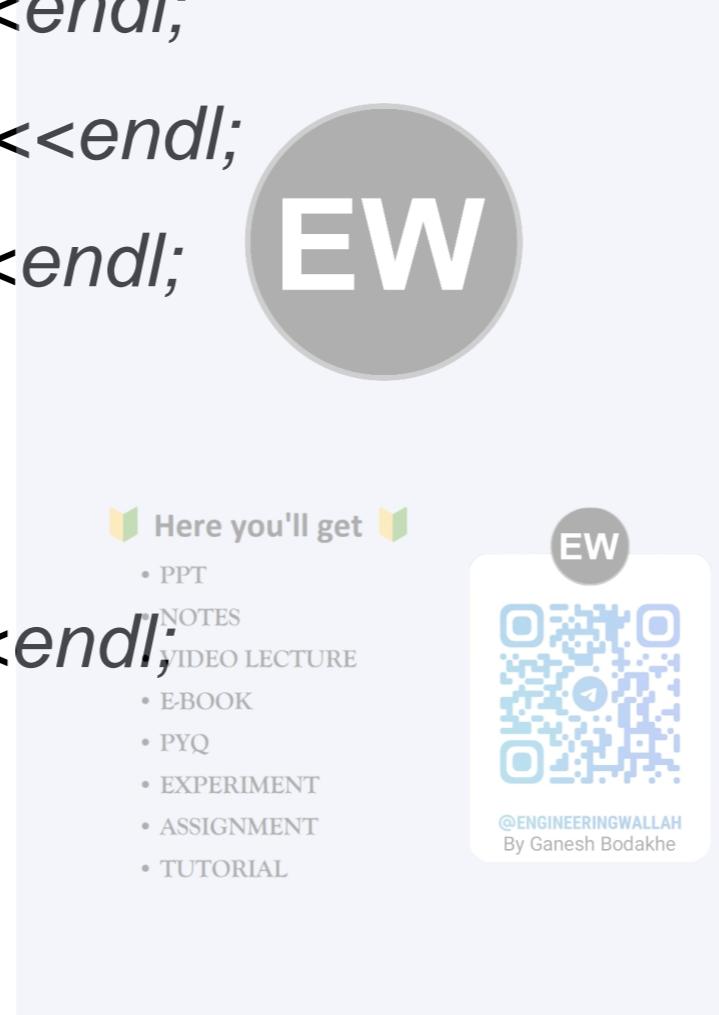
Stack operation using linked list

```
void display() {  
    struct Node* ptr;  
  
    if(top==NULL)  
        cout<<"stack is empty";  
  
    else {  
        ptr = top;  
  
        cout<<"Stack elements are: ";  
  
        while (ptr != NULL) {  
            cout<< ptr->data << " ";  
  
            ptr = ptr->next;  
        }  
    }  
  
    cout<<endl;  
}
```



Stack operation using linked list

```
int main() {  
    int ch, val;  
  
    cout<<"1) Push in stack"<<endl;  
    cout<<"2) Pop from stack"<<endl;  
    cout<<"3) Display stack"<<endl;  
    cout<<"4) Exit"<<endl;  
  
    do {  
        cout<<"Enter choice: "<<endl;  
        cin>>ch;  
  
        switch(ch) {  
            case 1: {  
                cout<<"Enter value to be pushed:"<<endl;  
                cin>>val;  
                push(val);  
            } break;  
        }  
    } while(ch != 4);  
}  
cout<<"Stack is empty!"<<endl;
```



Stack operation using linked list

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

cout<<"Exit"<<endl;

break;

}

default: {

cout<<"Invalid Choice"<<endl;

}

}

EW }
 while(ch!=4);

return 0;

Here you'll get }

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

EW

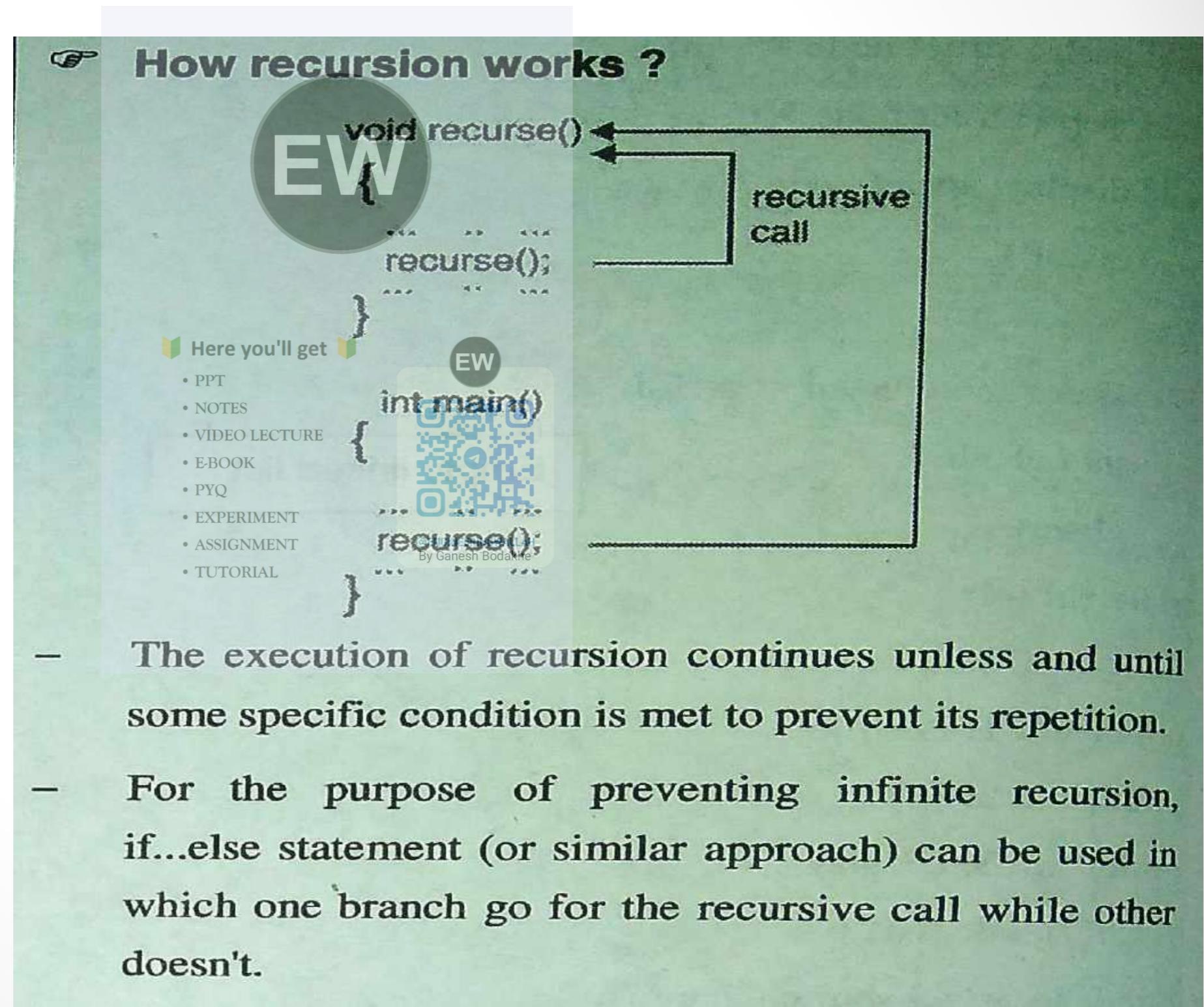


@ENGINEERINGWALLAH
By Ganesh Bodakhe

RECURSION IN STACK

“Calling function inside itself is called as recursion. Such function is called as recursive function”

How recursion works?



RECURSION IN STACK

Advantages :

1. It helps to reduce size of program
2. Easy to maintain function calling
3. Evaluation of stack can be through recursion

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL

EW



@ENGINEERINGWALLAH

By Ganesh Bedamne

Disadvantages :

1. It takes more time bcz of stack overlapping
2. Stack overflow may occur
3. Memory requirement is more
4. Efficiency is less

Backtracking algorithm strategy:

BACKTRACKING

- Backtracking is a simple, elegant, recursive technique which can be put to a variety of uses.
- You start at the root of a tree, the tree probably has some good and bad leaves. You want to get to a good leaf. At each node, you choose one of its children to move to, and you keep this up in a stack until you get to a leaf.
- Suppose you get to a bad leaf. You can *backtrack* to continue the search for a good leaf by revoking your *most recent* choice, and trying out the next option in that set of options.
- If you run out of options, revoke the choice that got you here, and try another choice at that node.
- If you end up at the root with no options left, there are no good leaves to be found.



Here you'll get

• PPT

• NOTE

• VIDEO LECTURE

• E-BOOK

• PYO

• EXPERIMENT

• ASSIGNMENT

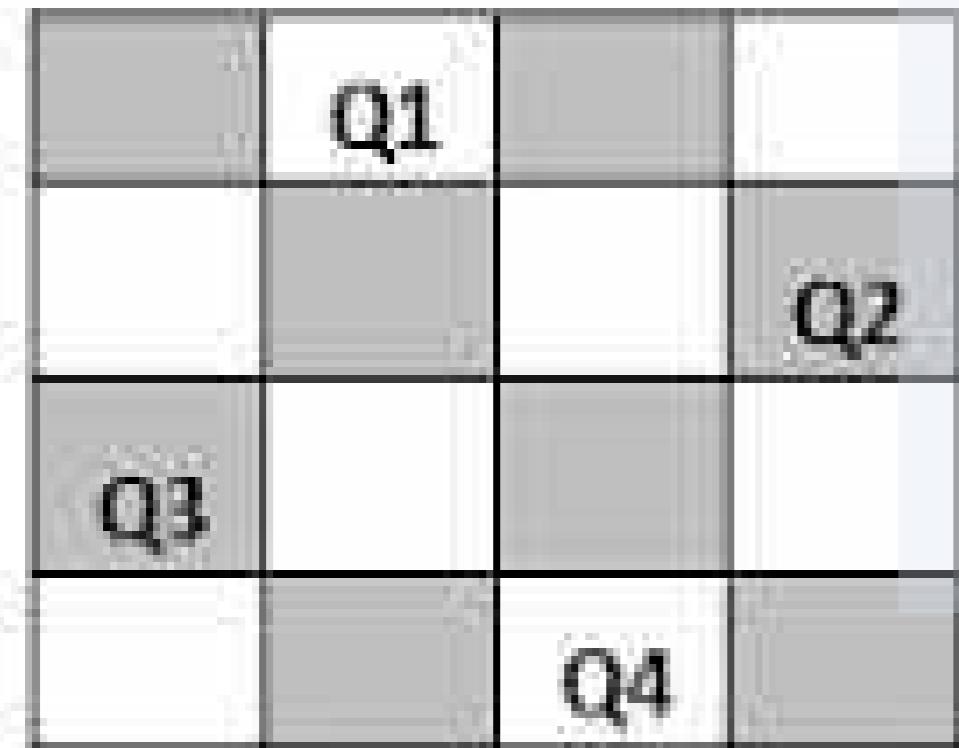
• TUTORIAL

EW

@ENGINEERINGWALLAH
By Ganesh Bodakhe

4-QUEEN PROBLEM

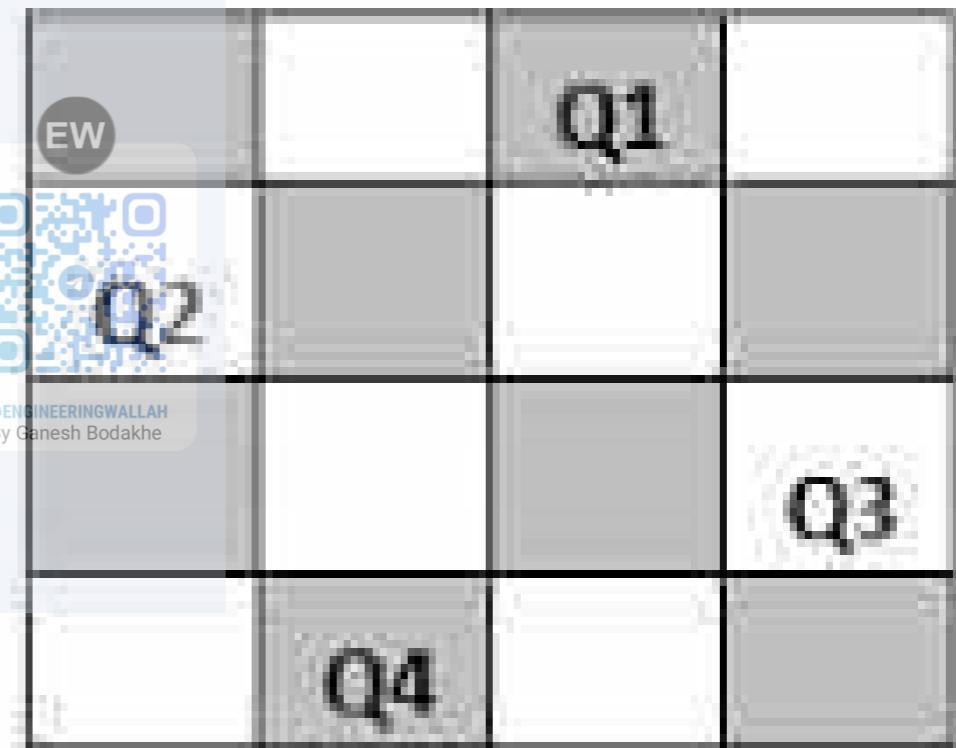
The N Queen is the problem of placing N chess queens on an NxN chessboard so that no two queens attack each other.



Solution 1

Here you'll get

- PPT
- NOTES
- VIDEO LECTURE
- E-BOOK
- PYQ
- EXPERIMENT
- ASSIGNMENT
- TUTORIAL



Solution 2