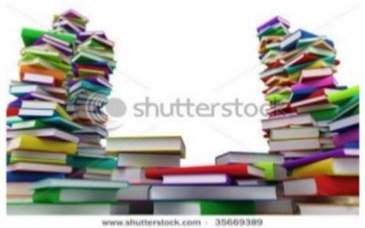
Algorithms & Data Structure

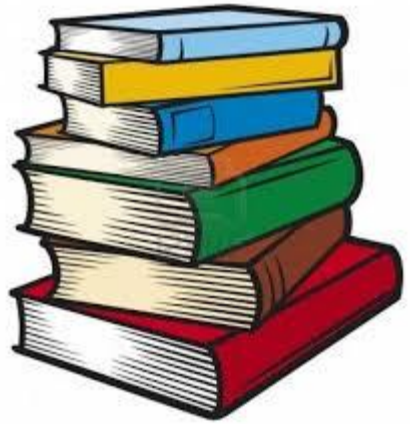
Kiran Waghmare

Examples of stack



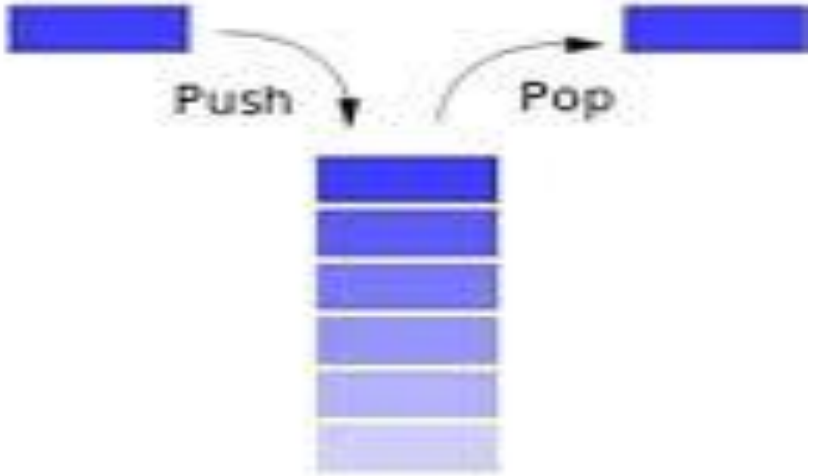
Stacks

Kiran Waghmare



Stack of books

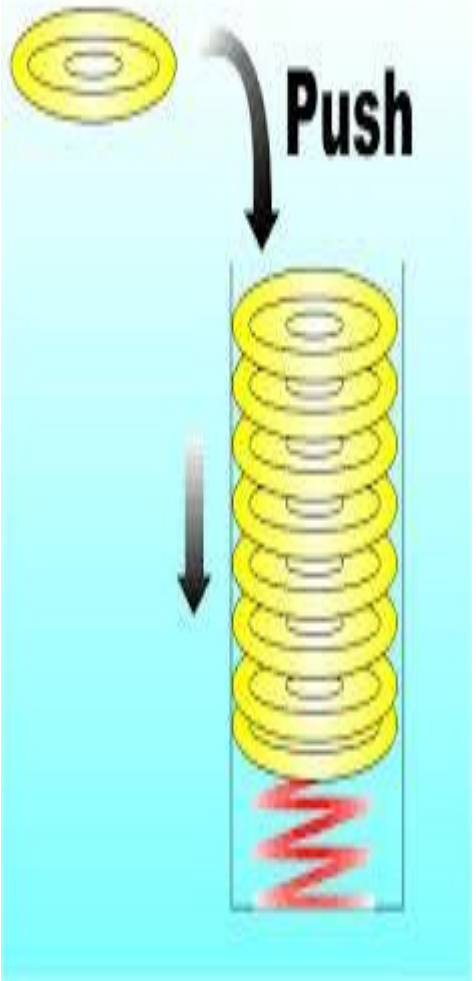
Stack of Coins



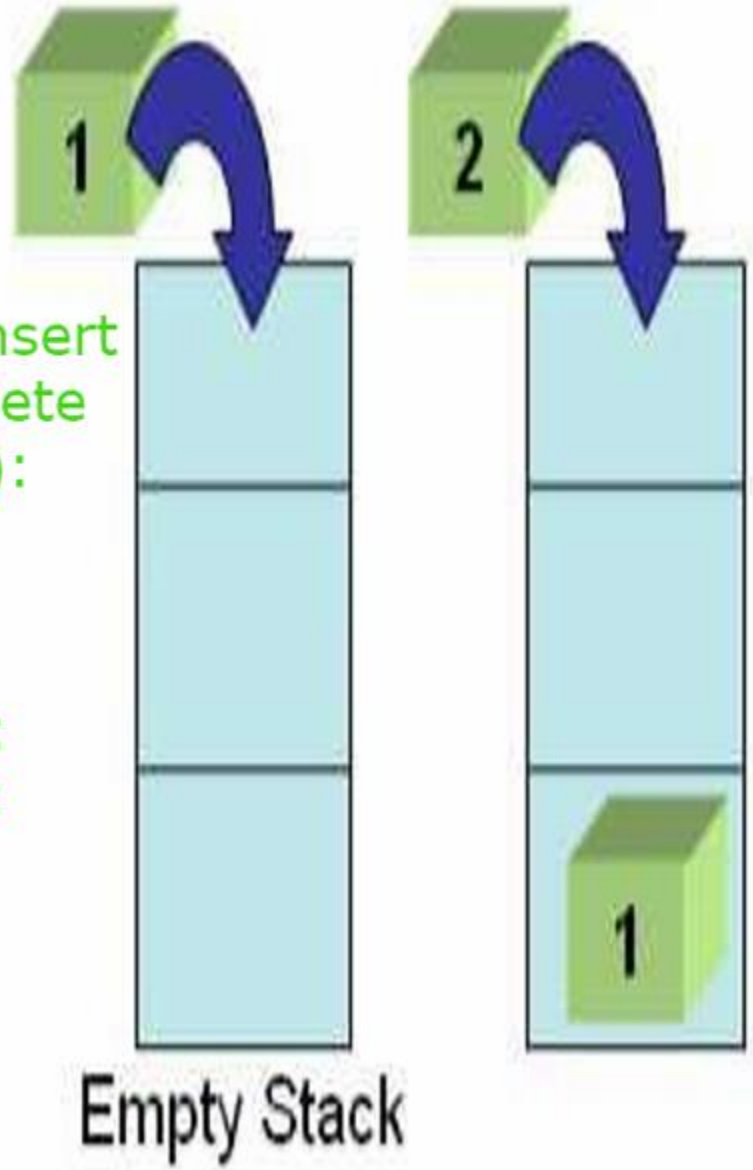
Memory stack

Stack

- Stack is an ordered list of similar data type.
- 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.



PUSH : Insert
POP : Delete
isEmpty():
isFull():
peek():
count():
change():
display():



Who can see what you share here? Recording On

10 20

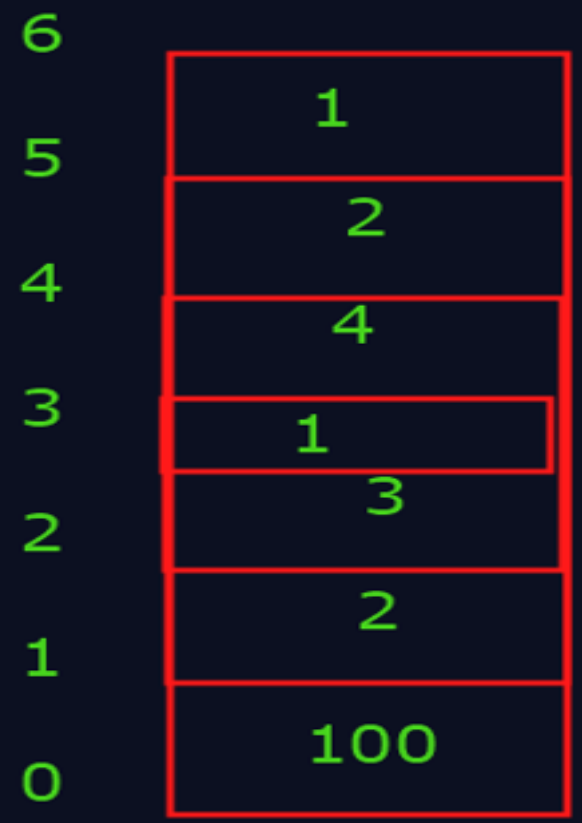
20 TOP
0 10 TOS

Talking: Kiran Waghmare

2
1

the
first object to
come out.

Array Representation of Stack




```
int S[7];
TOS=30
TOS=-1;
push(10);
push(20);
Push(30);
Push(40);
Pop();
pop(20);
pop();
pop();
```

Underflow:stack is empty & we are trying to delete element

Overflow: Stack is full & we are trying to insert an element

Standard Stack Operations

- The following are some common operations implemented on the stack:
- push(): When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.

- pop(): When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.

- isEmpty(): It determines whether the stack is empty or not.
- isFull(): It determines whether the stack is full or not.'

- peek(): It returns the element at the given position.
- count(): It returns the total number of elements available in a stack.
- change(): It changes the element at the given position.

- display(): It prints all the elements available in the stack.

Steps:

1. **If** TOP ≥ SIZE **then**
2. + **Print** "Stack is full"
3. **Else**
4. TOP = TOP + 1
5. A[TOP] = ITEM
6. **EndIf**
7. **Stop**

PUSH Operation Algorithm

Steps:

1. **If** TOP < 1 **then**
2. **Print** "Stack is empty"
3. **Else**
4. ITEM = A[TOP]
5. TOP = TOP – 1
6. **EndIf**
7. **Stop**

POP Operation Algorithm

Steps:

1. **If** TOP ≥ SIZE **then**
2. + **Print** "Stack is full"
3. **Else**
4. TOP = TOP + 1
5. A[TOP] = ITEM
6. **EndIf**
7. **Stop**

static

PUSH Operation Algorithm

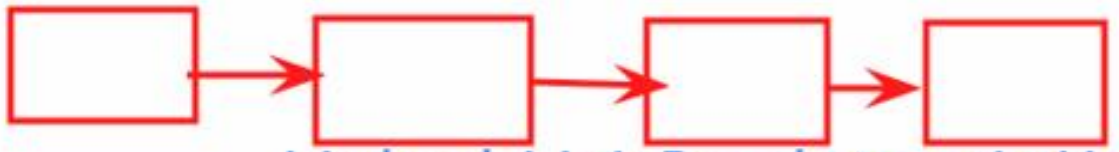
Array Implementation

Steps:

1. **If** TOP < 1 **then**
2. **Print** "Stack is empty"
3. **Else**
4. ITEM = A[TOP]
5. TOP = TOP – 1
6. **EndIf**
7. **Stop**

Dynamic

POP Operation Algorithm



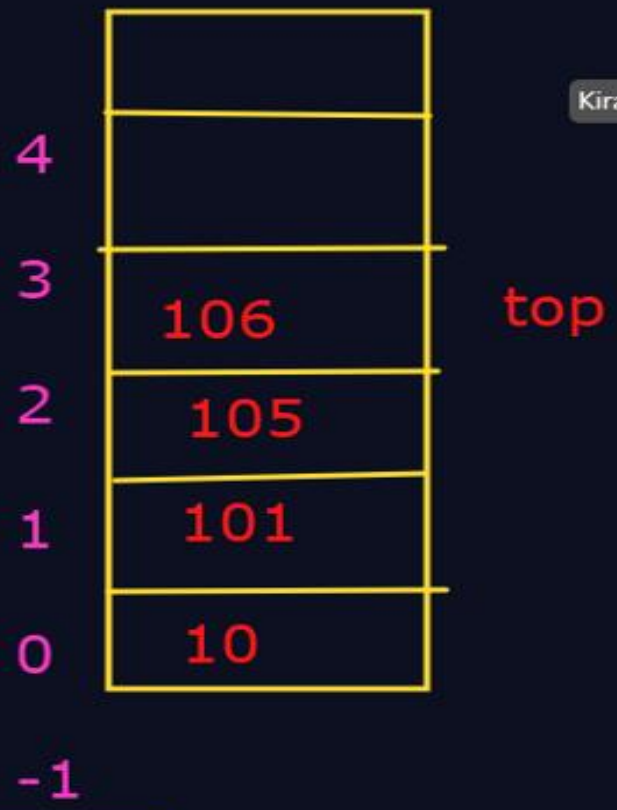
Linked List Implementation

```
class Stack
{
    static final int MAX = 5;
    int top;
    int a1[] =new int[MAX];

    Stack()
    {
        top=-1;
    }

    boolean isEmpty()
    {
        return (top < 0);
    }

    boolean isFull()
    {
        return (top > (MAX-1));
    }

    boolean push(int x)
    {
        if(top >= (MAX-1))
        {
            System.out.println("Overflow !!!!");
            return false;
        }
        else
        {
            a1[++top] = x;
            //top=top+1;=>preincrement
            //a1[top] = x;
            System.out.println("Push :"+x);
            return true;
        }
    }

    int pop()
    {
        if(top < 0 )
        {
            System.out.println("Underflow !!!");
            return 0;
        }
        else
```

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo

Who can see what you share here? Recording On

Kiran Wa



stack

StackApp

```
}

class StackApp
{
    public static void main(String args
    {
        Stack1 s = new Stack1(5);

        s.push(11);
        s.push(22);
        s.push(33);
        s.display();
        s.pop();
        s.display();
        s.push(55);
        s.push(3);
        s.display();
```

Who can see what you share here? Recording On

Command Prompt

Kiran Waghm...

```
0
0

C:\Test>javac StackApp.java

C:\Test>java StackApp
11
22
33
0
0
11
22
33
0
0
11
22
55
3
0

C:\Test>
```

```
public void push(long x)
{
    S1[++top] = x;
}

public long pop()
{
    return S1[top--];
}

public long peek()
{
    return S1[top];
}

public boolean isEmpty()
{
    return (top == -1);
}
```


RADHESHAM

M
A
H
S
E
H
D
A
R

MAHSEHDAR

Applications of Stack

- The following are the applications of the stack:

- Balancing of symbols
- String reversal
- UNDO/REDO
- Recursion
- DFS(Depth First Search)
- Backtracking
- UNDO/REDO
- Recursion
- DFS(Depth First Search)
- Backtracking
- Expression conversion
- Memory management

Polish Notations

a+b*c/e

a,b,c : operand
+,-, *, / : Operator

Infix: a+b

Prefix: +ab

Postfix: ab+

Infix to Postfix Conversion

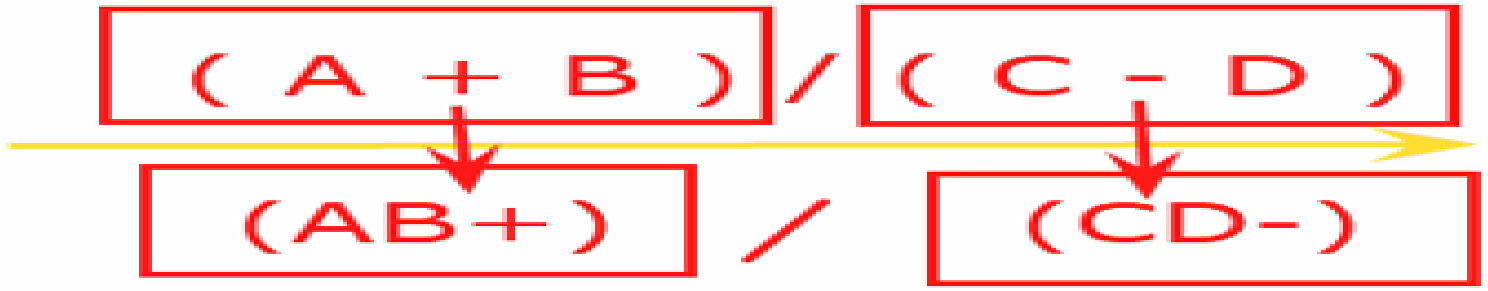
Infix to Pre-fix Conversion

Postfix Expression Evaluation

Precedence and associativity of operators

Operators	*Precedence*	*Associativity*
– (unary), +(unary), NOT	6	–
^ (exponentiation)	6	Right to left
* (multiplication), / (division)	5	Left to right
+ (addition), – (subtraction)	4	Left to right
<, <=, +, < >, >=	3	Left to right
AND	2	Left to right
OR, XOR	1	Left to right

Polish Notations



Postfix: AB+CD-/

A+(B/C)-D

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

A+[(B+C)+(D+E)*F]/G

Expression Evaluation

- A + B * C / D – E ^ F * G

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

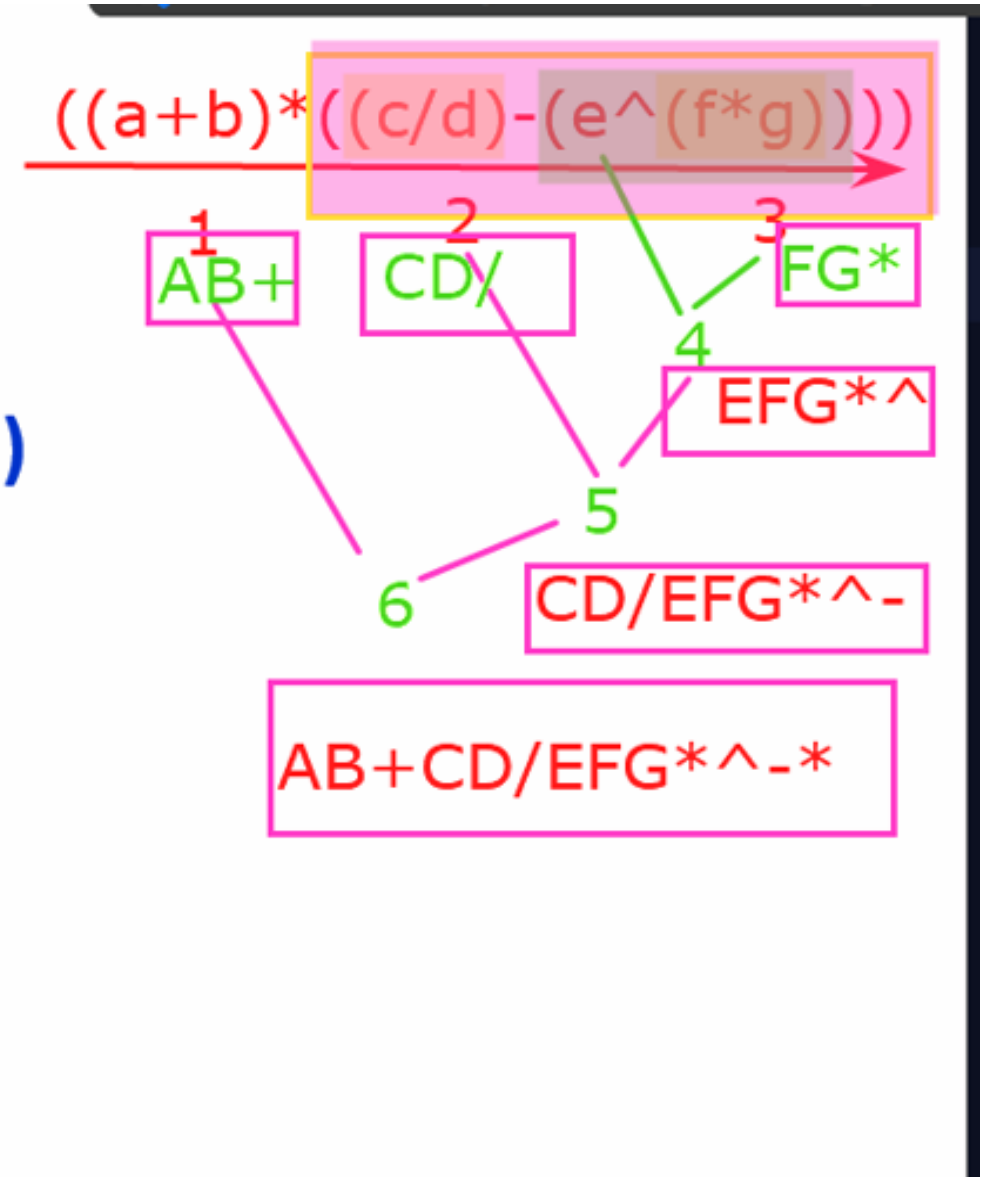
- (A + ((B ^ C) – D)) * (E - (A / C)))

Expression Evaluation

- A + B * C / D – E ^ F * G

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

- (A + ((B ^ C) – D)) * (E - (A / C)))



A+B-C

(AB+)-C

AB+C-

Conversion of Infix to Postfix Expression

Steps:

```
1.  TOP = 0, PUSH('(')              // Initialize the stack
2.  While (TOP > 0) do
3.      item = E.ReadSymbol( )      // Scan the next symbol in infix expression
4.      x = POP( )                  // Get the next item from the stack
5.    Case: item = operand          // If the symbol is an operand
6.        PUSH(x)                   // The stack will remain same
7.        Output(item)              // Add the symbol into the output expression
8.    Case: item = ')',             // Scan reaches to its end
9.        While x ≠ '(' do          // Till the left match is not found
10.         Output(x)
11.         x = POP( )
12.     EndWhile
```

Example: Convert X: A+(B*C-(D/E)*G)*H into Postfix form showing stack status after every step in tabular form.

Solution: Expression: x := (A+(B*C-(D/E)*G)*H))

Symbol scanned	Stack	Postfix Expression
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
)	(+(-	ABC*DE/
*	(+(-*	ABC*DE/
G	(+(-*	ABC*DE/G
)	(+	ABC*DE/G*-
*	(+*	ABC*DE/G*-
H	(+*	ABC*DE/G*-H
)		ABC*DE/G*-H*+
$		

ALGORITHM:

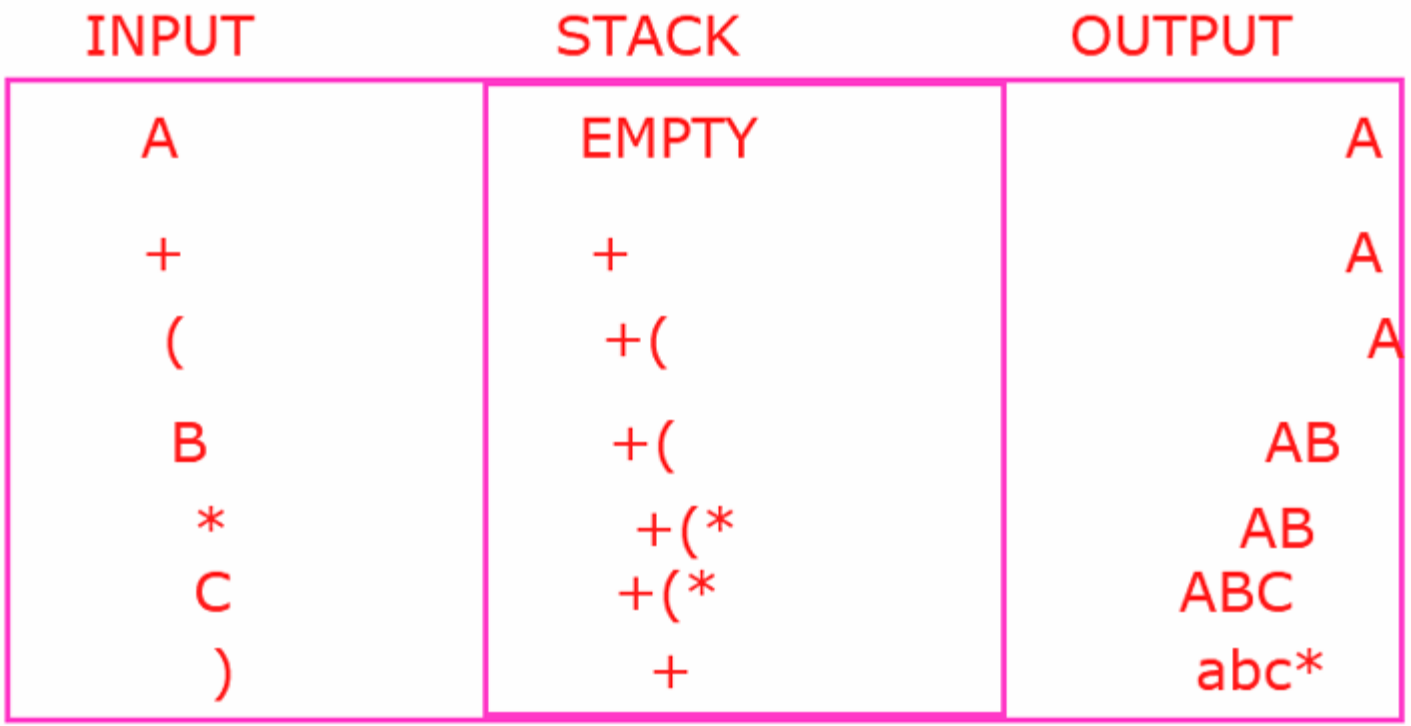
- **Scan infix expression from left to right.**
- **If there is a character as operand, output it.**
- **if not**

  1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.

  2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
- **If the scanned character is an '(', push it to the stack.**
- **If the character character is an ')', pop the stack and and output it until a '(' is encountered, and discard both the parenthesis.**
- **Repeat steps 2-6 until infix expression is scanned.**
- **display the output**
- **Pop and output from the stack until it is not empty.**

Polish Notations

A+(B*C)



INPUT	STACK	OUTPUT
A	EMPTY	A
+	+	A
(	+(	A
B	+(	AB
*	+(*	AB
C	+(*	ABC
)	+	abc*

ABC*+

Evaluation of Postfix Expression

Steps:

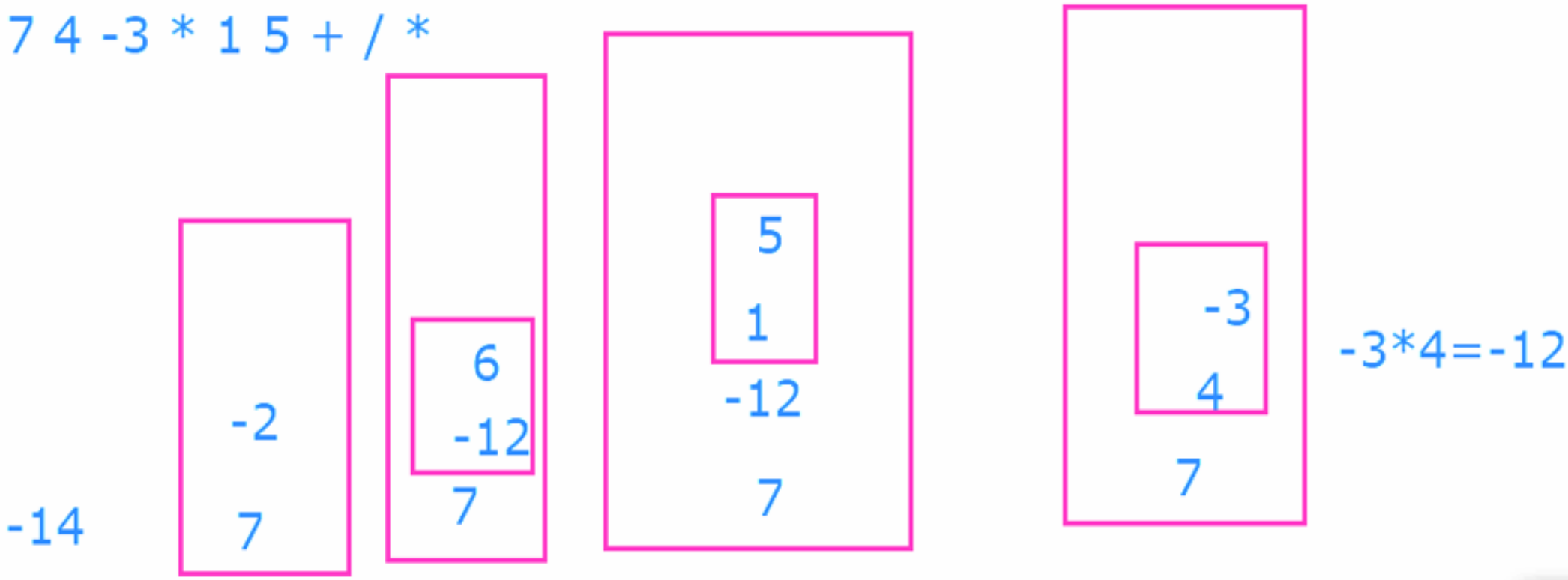
1. Append a special delimiter '#' at the end of the expression
2. item = *E*.**ReadSymbol( )** // Read the first symbol from *E*
3. **While** (item ≠ '#') **do**
4. **If** (item = operand) **then**
5. **PUSH**(item) // Operand is the first push into the stack
6. **Else**
7. op = item // The item is an operator
8. *y* = **POP( )** // The right-most operand of the current operator
9. *x* = **POP( )** // The left-most operand of the current operator
10. *t* = *x* op *y* // Perform the operation with operator 'op' and operands *x*, *y*
11. **PUSH**(*t*) // Push the result into stack
12. **EndIf**
13. item = *E*.**ReadSymbol( )** // Read the next item from *E*
14. **EndWhile**
15. value = **POP( )** // Get the value of the expression
16. **Return**(value)
17. **Stop**

Polish Notations

POSTFIX EVALUATION

3 10 5 + *

7 4 -3 * 1 5 + / *



Thanks