Algorithms & Data Structure

Kiran Waghmare

==========

Day 8 : 27/04/2022

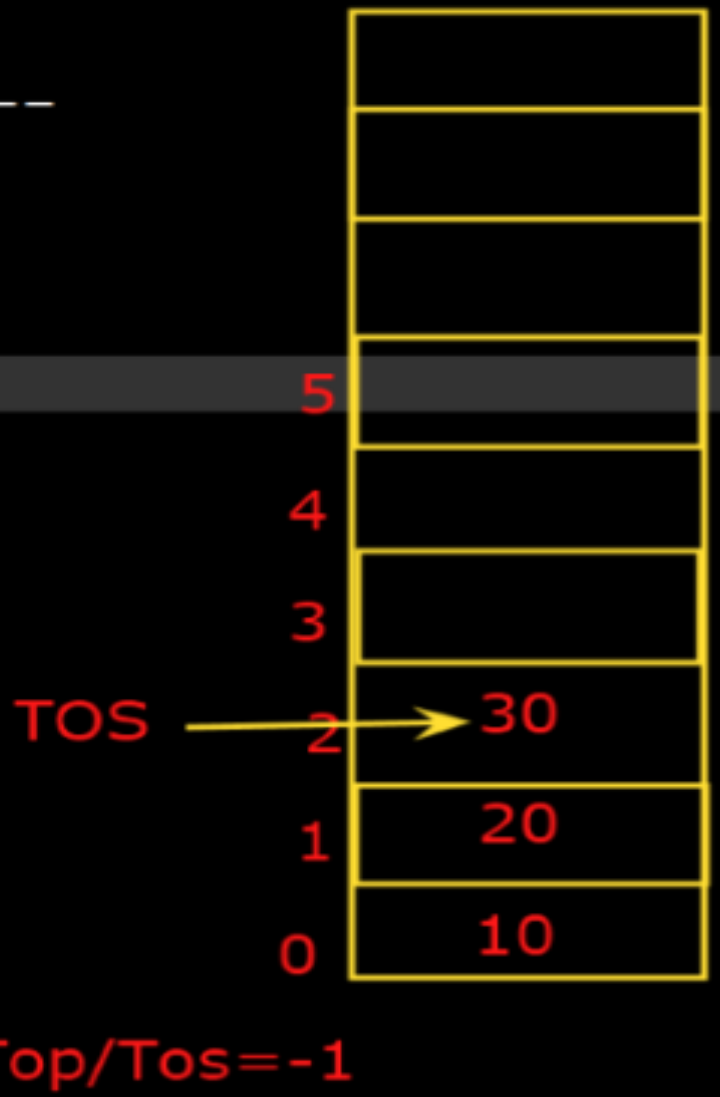
Topic : Stack

Stack:

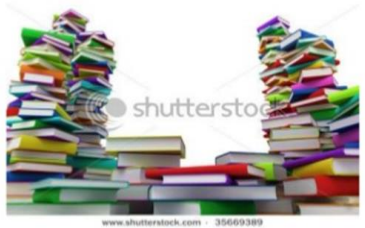
-ordered data structure

-Top/Tos

-TOS:Top of Stack

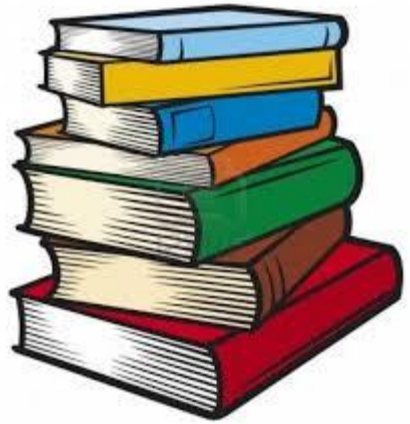


Examples of stack



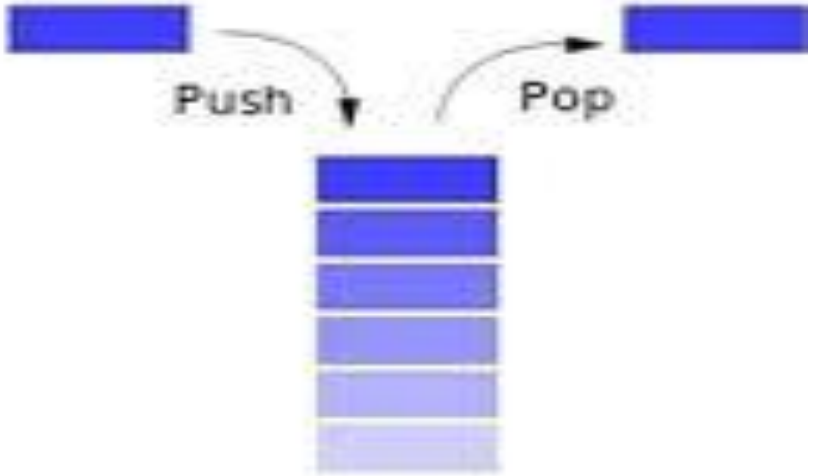
Stacks

Kiran Waghmare



Stack of books

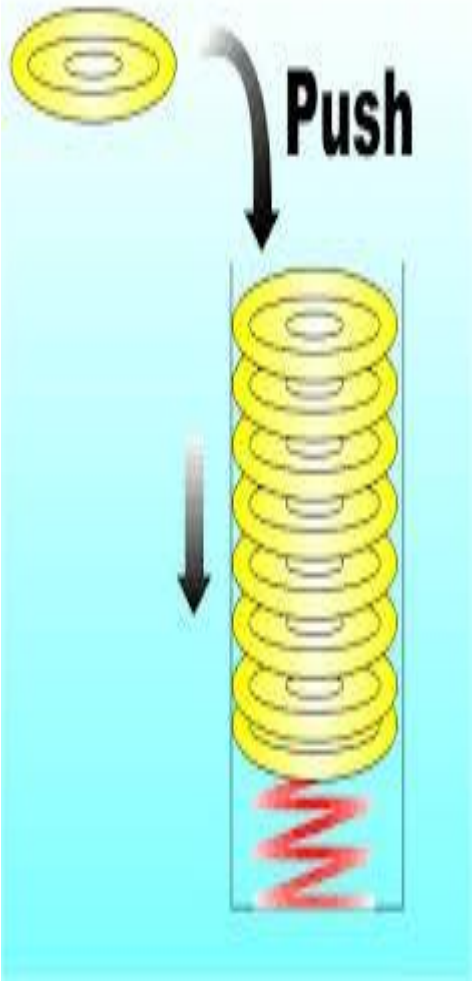
Stack of Coins



Memory stack

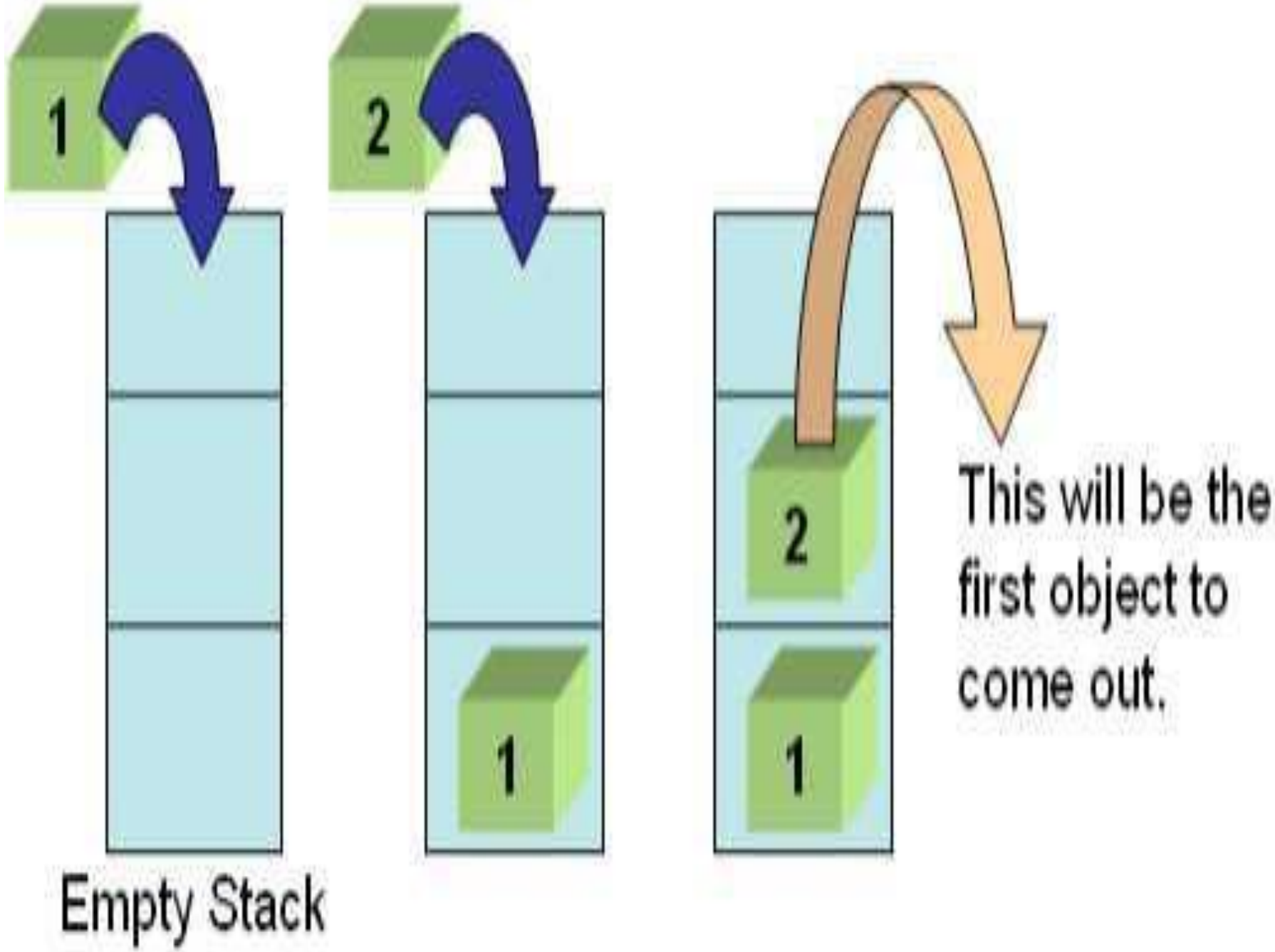
Stack

- 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.



Standard Stack Operations

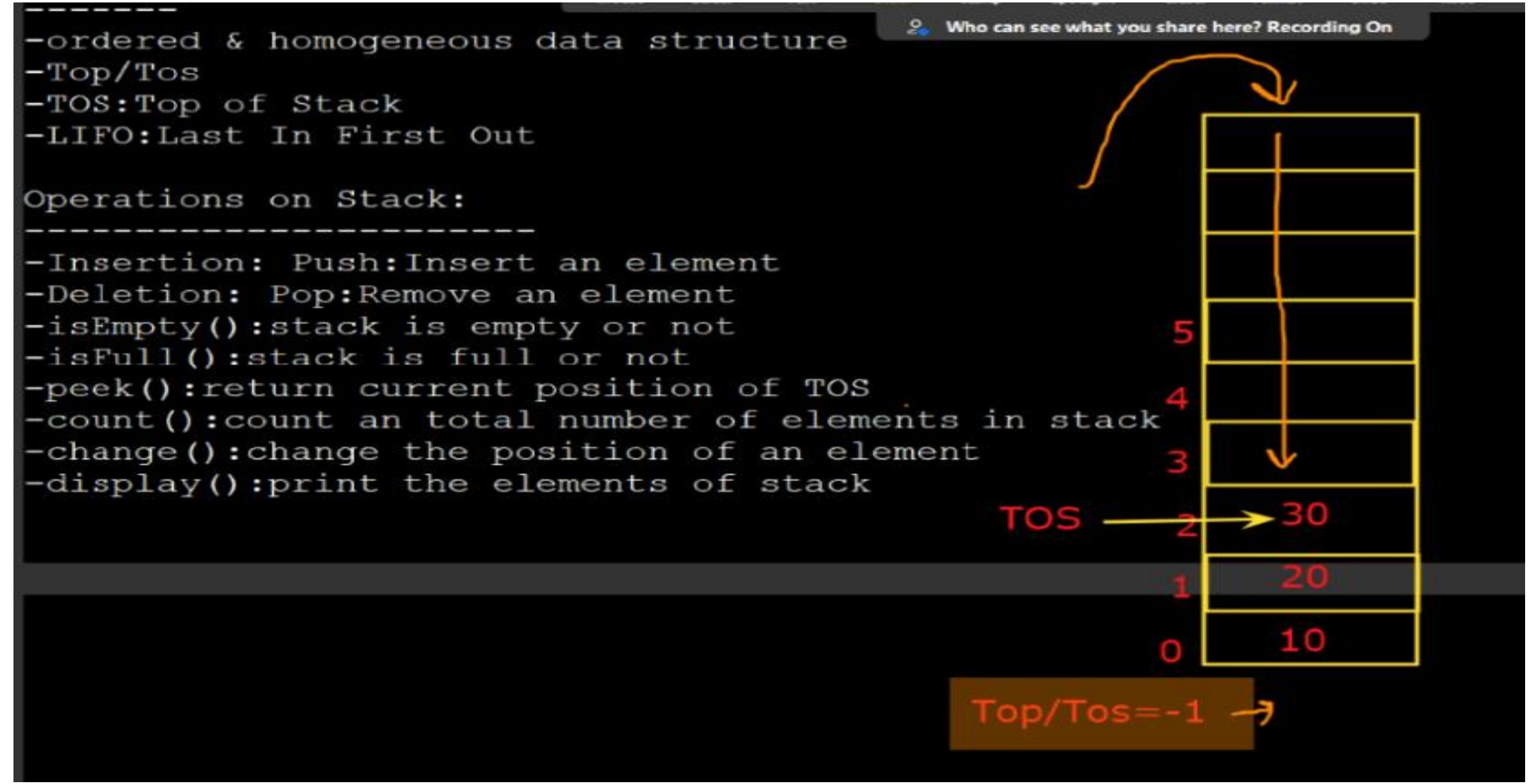
- **The following are some common operations implemented on the stack:**
- **push():**
  - When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():**
  - When we delete an element from the stack, the operation is known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

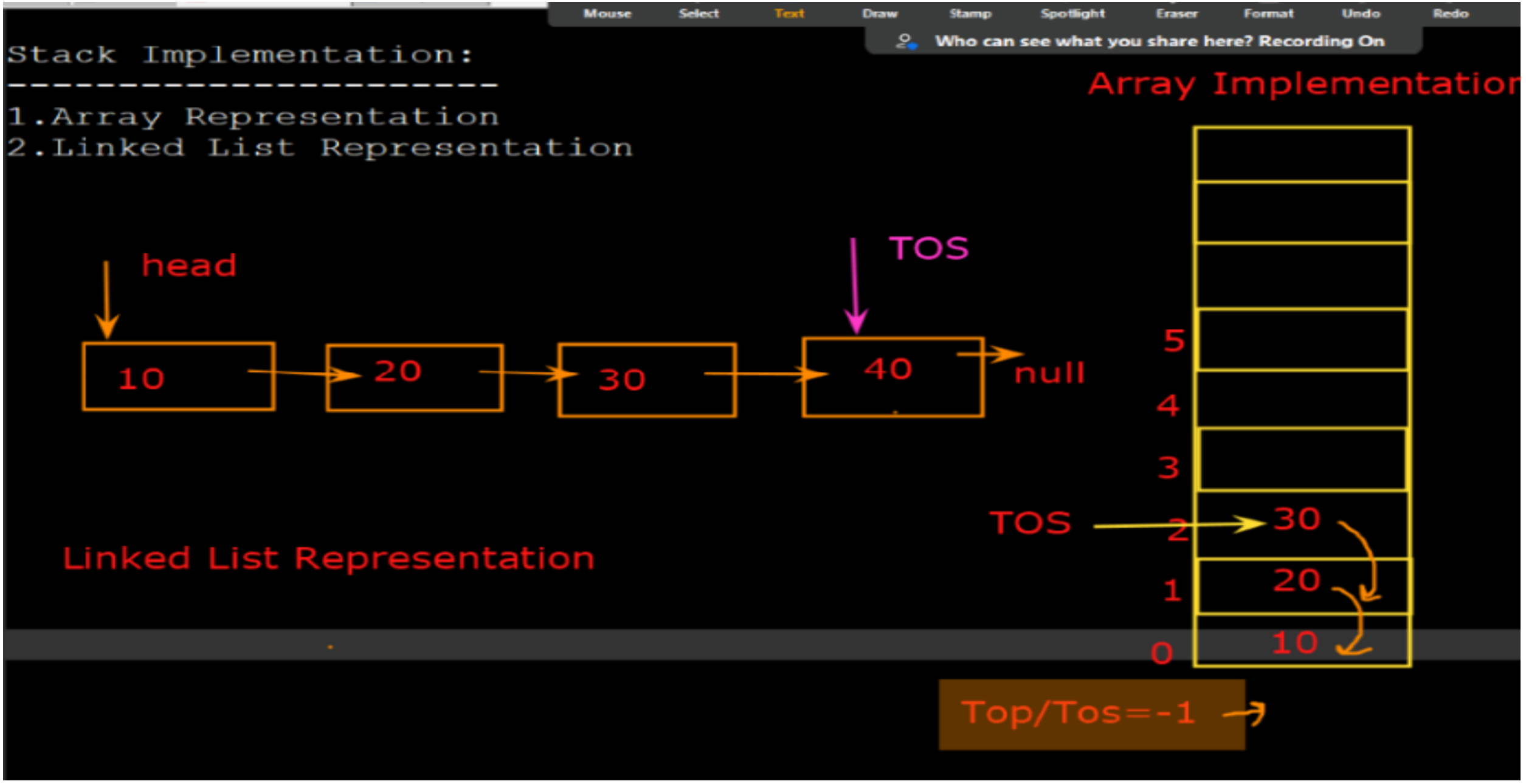

1
2
2
1
2
1
This will be the first object to come out.
Empty Stack

-ordered & homogeneous data structure
-Top/Tos
-TOS:Top of Stack
-LIFO:Last In First Out

Operations on Stack:

-Insertion: Push:Insert an element
-Deletion: Pop:Remove an element
-isEmpty():stack is empty or not
-isFull():stack is full or not
-peek():return current position of TOS
-count():count an total number of elements in stack
-change():change the position of an element
-display():print the elements of stack




Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Who can see what you share here? Recording On
Stack Implementation:
1.Array Representation
2.Linked List Representation
Array Implementation
head
TOS
10
20
30
40
null
Linked List Representation
TOS
5
4
3
2
1
0
30
20
10
Top/Tos=-1

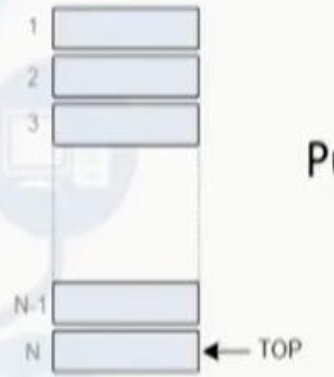
Operation: Push with array

We have assumed that the array index varies from 1 to SIZE and TOP points the location of the current top-most item in the stack. The following algorithm defines the insertion of an item into a stack represented using an array A.

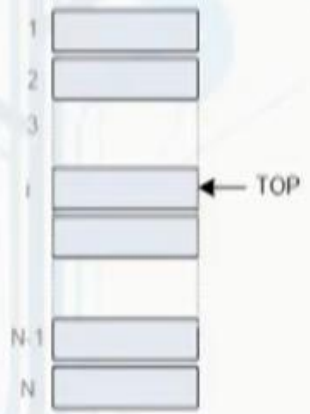
Input: The new item ITEM to be pushed onto it.
Output: A stack with a newly pushed ITEM at the TOP position.
Data structure: An array *A* with TOP as the pointer.

Steps:

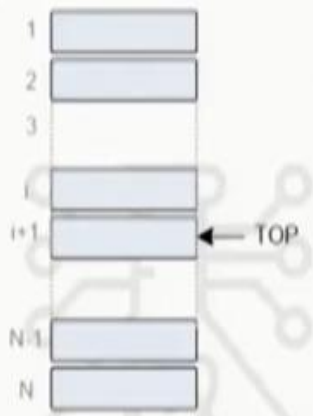
1. **If** TOP ≥ SIZE **then**
2. + **Print** "Stack is full"
3. **Else**
4. TOP = TOP + 1
5. A[TOP] = ITEM
6. **EndIf**
7. **Stop**



Push operation is failed



Before push()



After push()

Insertion in Stack

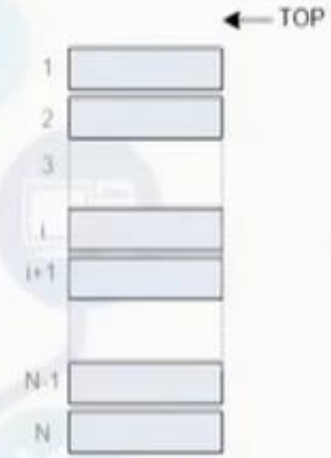
Operation: Pop with array

The following algorithm defines the deletion of an item from a stack represented using an array A.

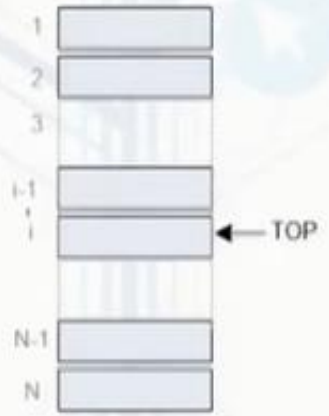
Input: A stack with elements.
Output: Removes an ITEM from the top of the stack if it is not empty.
Data structure: An array A with TOP as the pointer.

Steps:

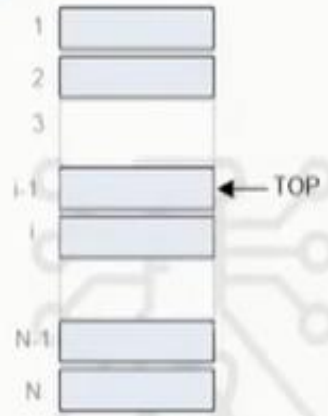
1. **If** TOP < 1 **then**
2. **Print** "Stack is empty"
3. **Else**
4. ITEM = A[TOP]
5. TOP = TOP – 1
6. **EndIf**
7. **Stop**



Pop operation is failed



Before Pop()



After Pop()

Deletion in Stack

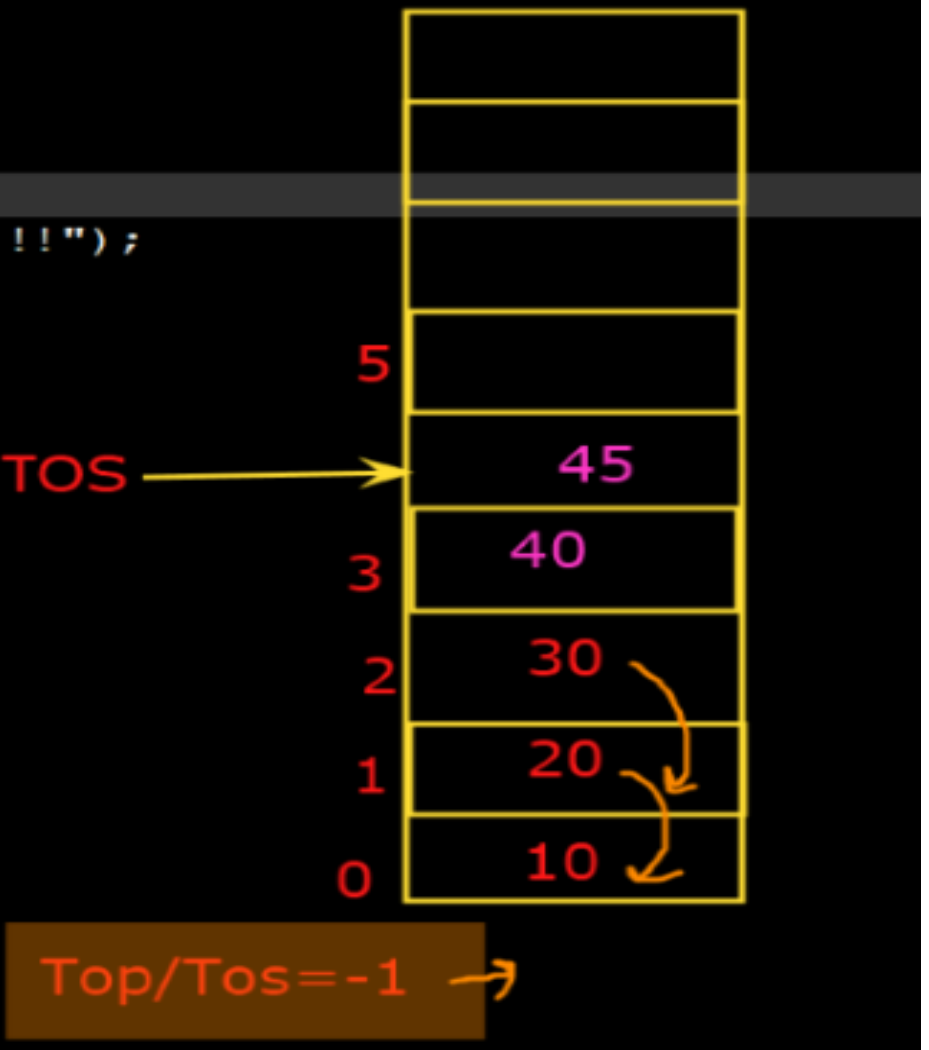
```
boolean push(int x)
{
    if(top >= Max-1){
        System.out.println("Overflow !!!");
        return false;
    }
    else {
        s[++top] = x;
        System.out.println(x+"----> Push operation!!!");
        return true;

    }

}
```

Pop Operation:

```
int pop()
{
    if(top < 0){
    System.out.println("Underflow !!!");
    return 0;
    }
    else{
        int x = s[top--];
        return x;
    }
}
```



-String reversal:

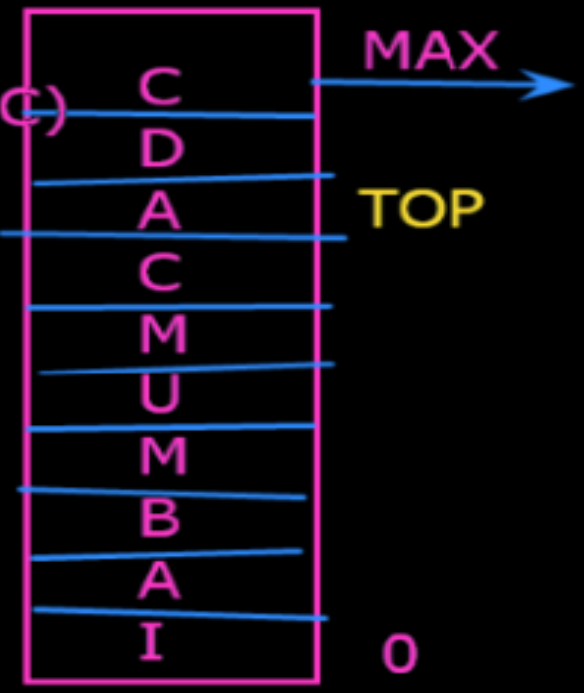
Who can see what you share here? Recording On

cdacmumbai

push(IABMUMCADC)

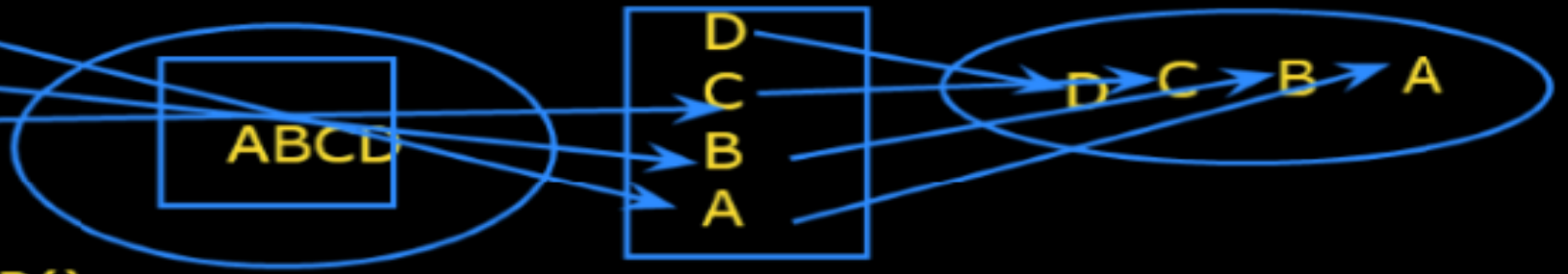
POP()

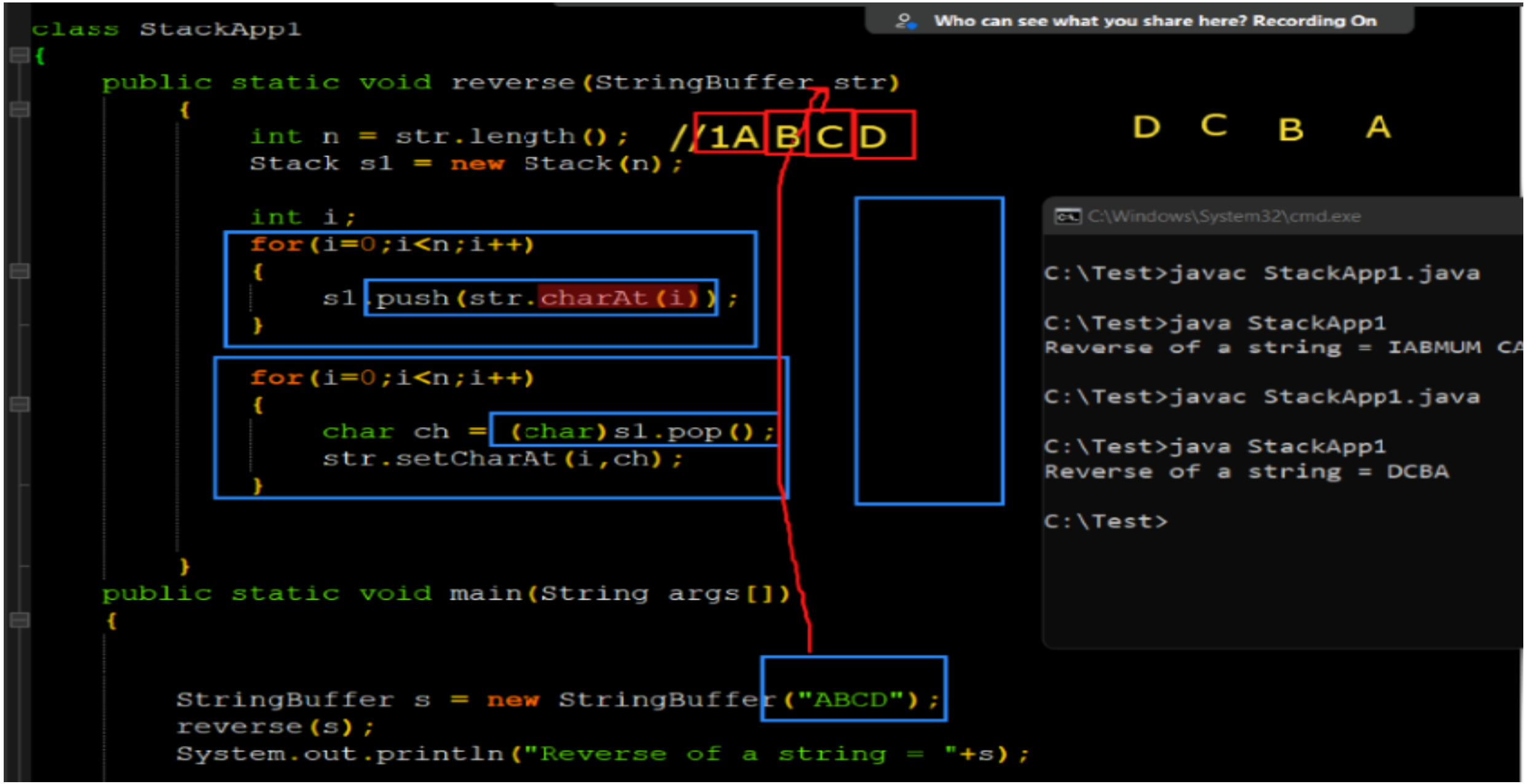
C D A C M



pUSH(A)
PUSH(B)
pUSH(C)
pUSH(D)

POP()
POP()
POP()
pOP()




Who can see what you share here? Recording On
class StackApp1
{
public static void reverse(StringBuffer str)
{
int n = str.length(); //1A B C D
Stack s1 = new Stack(n);
int i;
for(i=0;i<n;i++)
{
s1.push(str.charAt(i));
}
for(i=0;i<n;i++)
{
char ch = (char)s1.pop();
str.setCharAt(i,ch);
}
}
public static void main(String args[])
{
StringBuffer s = new StringBuffer("ABCD");
reverse(s);
System.out.println("Reverse of a string = "+s);
D C B A
C:\Windows\System32\cmd.exe
C:\Test>javac StackApp1.java
C:\Test>java StackApp1
Reverse of a string = IABMUM CA
C:\Test>javac StackApp1.java
C:\Test>java StackApp1
Reverse of a string = DCBA
C:\Test>

Who can see what you share here? Recording On

Data Structure:Stack

case 1:

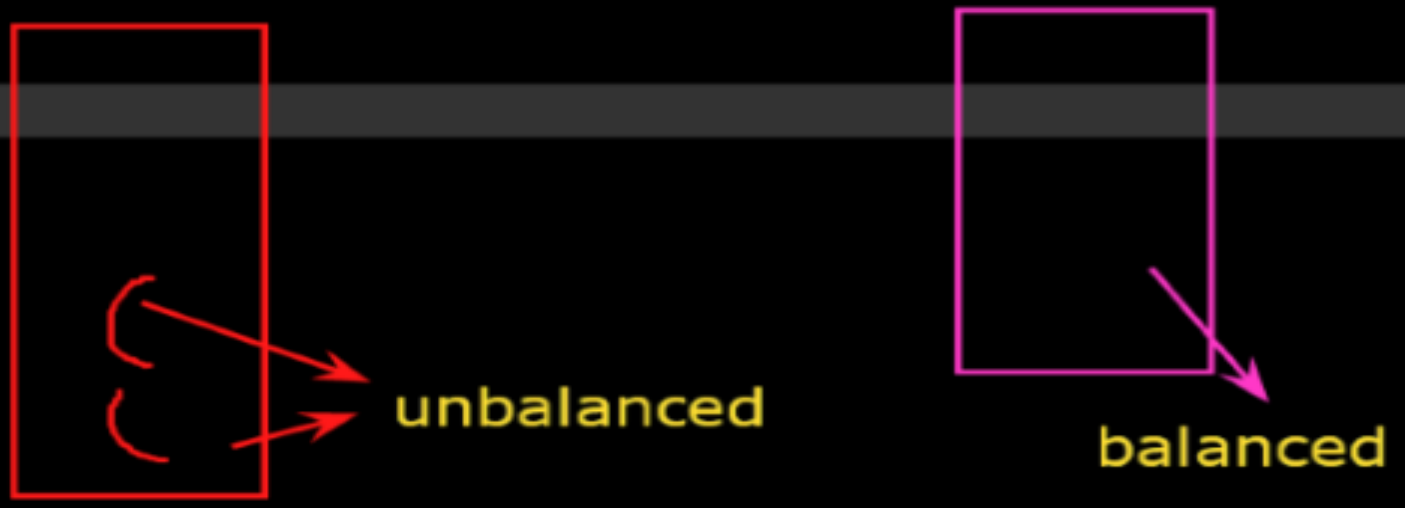
~~( ) ( ) ( )~~

NO. OF OPENING BRACKETS = NO OF CLOSING BRACKETS ==> Balanced

case 2:

~~(( ))~~ ((( ) :

NO. OF OPENING BRACKETS = NO OF CLOSING BRACKETS ==> UnBalanced



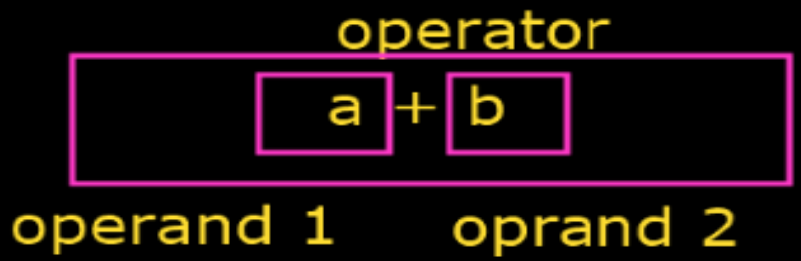
Polish Notations

1. **Infix Notation : A+B**
2. **Prefix Notation: +AB**
3. **Postfix Notation : AB+**

- **Operator Precedence:**
  1. **BODMAS Rule**
  2. **Brackets, Exponential, (* / %), (+ -)**
- **Rules: Infix to Postfix Conversion**
  1. Parenthesize the expression starting from left to right.
  2. During parenthesizing the expression, the operands associated with operator having higher precedence are first parenthesized. For example in above expression B* C is parenthesized first before A+B.
  3. The sub-expression (part of expression) which has been converted into postfix is to be treated as single operand.
  4. Once the expression is converted to postfix from remove the parenthesis.

Polish Notations:
---

1. Infix Notation
2. Prefix Notation
3. Postfix Notation



Infix Notation

a+b

Prefix Notation

+ab

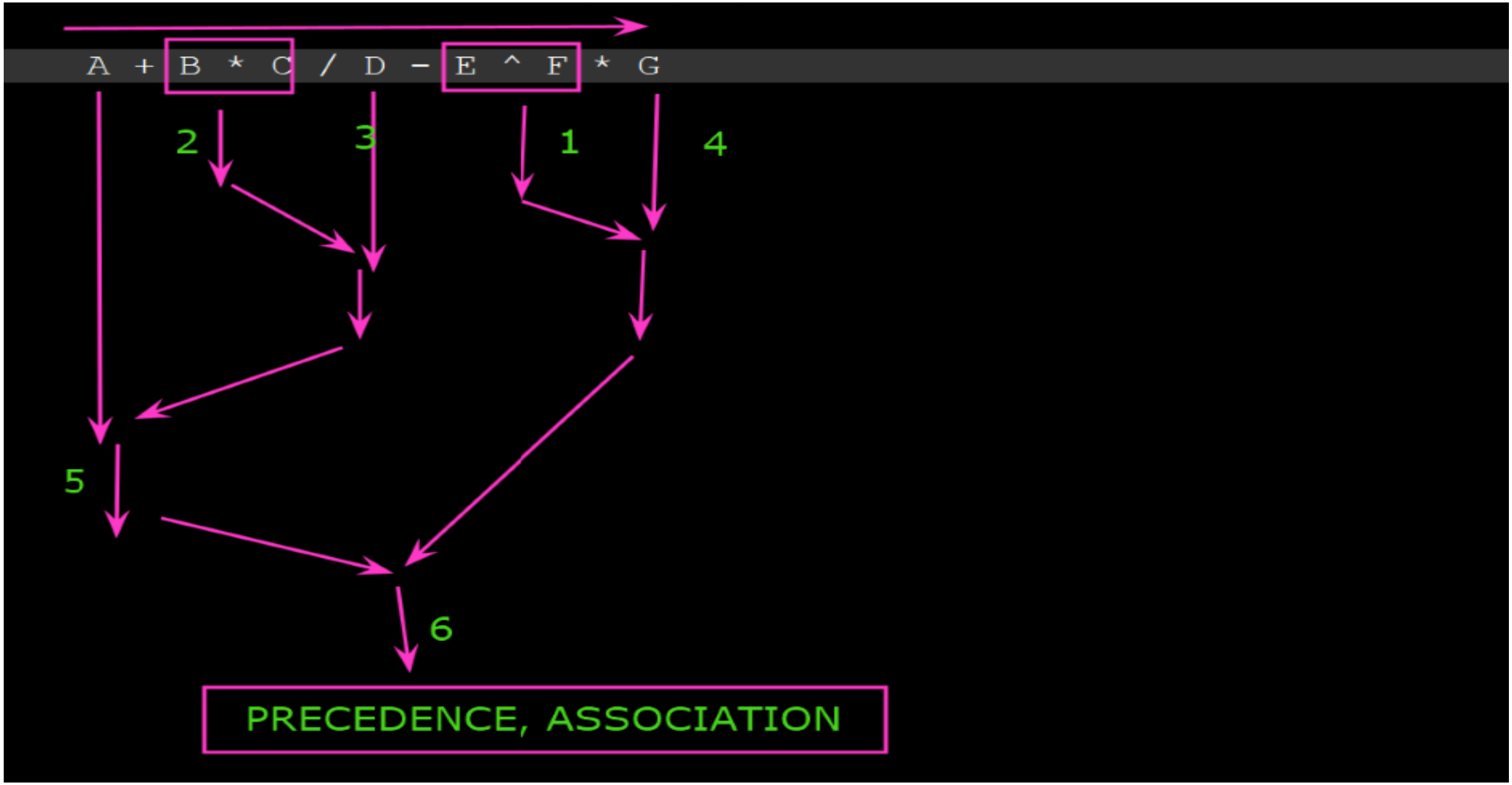
Postfix Notation

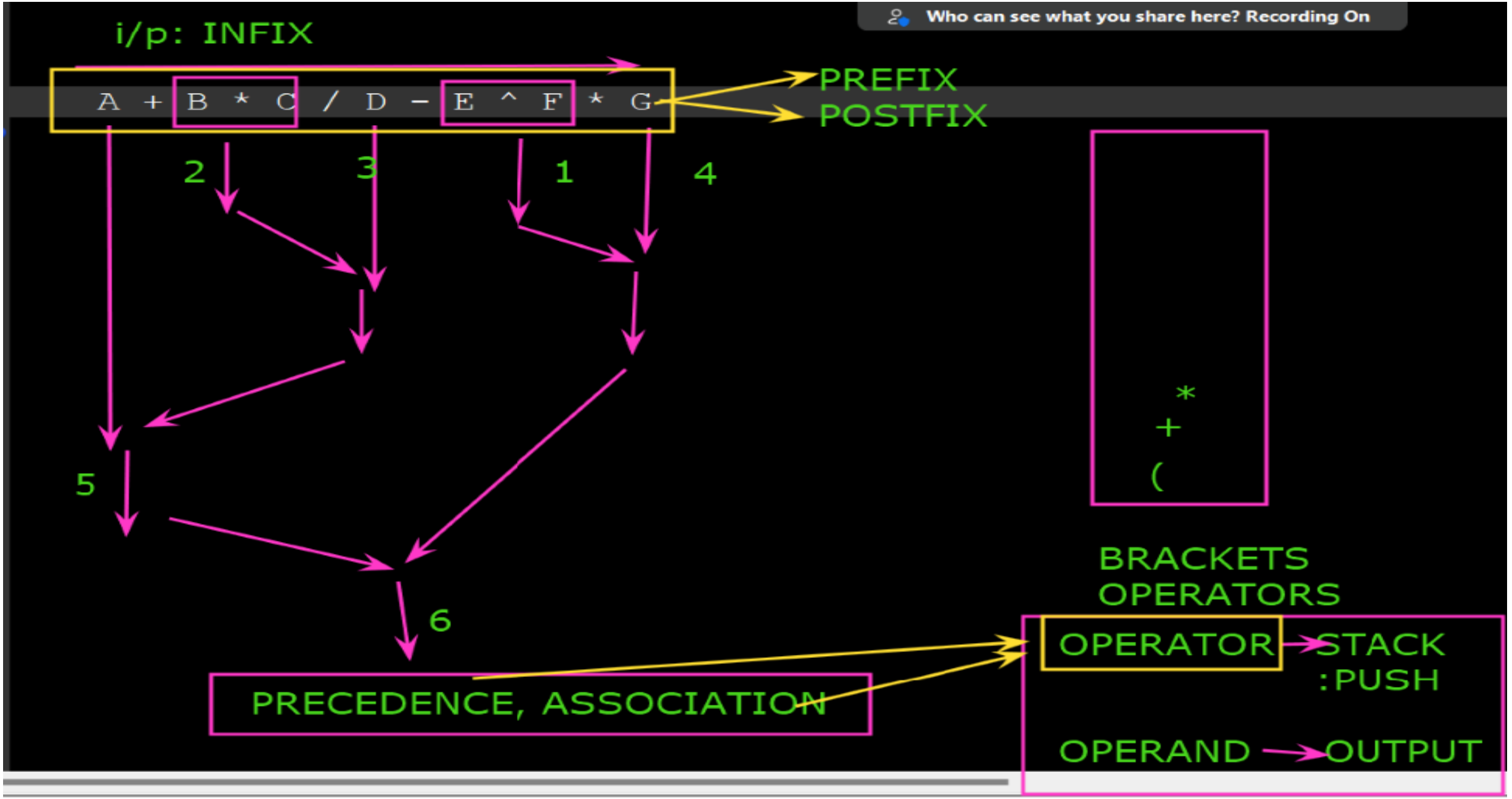
ab+

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

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


A + B * C / D - E ^ F * G
2
3
1
4
5
6
PRECEDENCE, ASSOCIATION


Who can see what you share here? Recording On
i/p: INFIX
A + B * C / D - E ^ F * G
PREFIX
POSTFIX
2
3
1
4
5
6
*
+
(
BRACKETS
OPERATORS
OPERATOR
STACK
:PUSH
OPERAND
OUTPUT
PRECEDENCE, ASSOCIATION

Application: Conversion of an infix expression to postfix expression

Input: E, simple arithmetic expression in infix notation delimited at the end by the right parenthesis ')', incoming and in-stack priority values for all possible symbols in an arithmetic expression.

Output: An arithmetic expression in postfix notation.

Data structure: Array representation of a stack with TOP as the pointer to the top-most element.

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
```

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.**

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo

Who can see what you share here? Recording On

Convert Infix to Postfix Notation usin

A + (B * C)

Character	Stack	Output
A		A
+	+	A
(	+(	A
B	+(	AB
*	+(*	AB
C	+(*	ABC
)	+	ABC*
-	-	ABC*+

PUSH:OPERATOR
HIGHER PRECEDENCE ELEMENT
POP: CLOSING BRACKET

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

EX: (A+(B*C-(D/E)*G)*H)

Character	Stack	Output
(	(	-
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+	ABC
)	(+	ABC*
/	(+/	ABC*
(	(+/(	ABC*
D	(+/(	ABC*D
-	(+/(-	ABC*D
E	(+/(-	ABC*DE
)	(+/	ABC*DE-
)	-	ABC*DE-/+

Application: Evaluation of a postfix expression

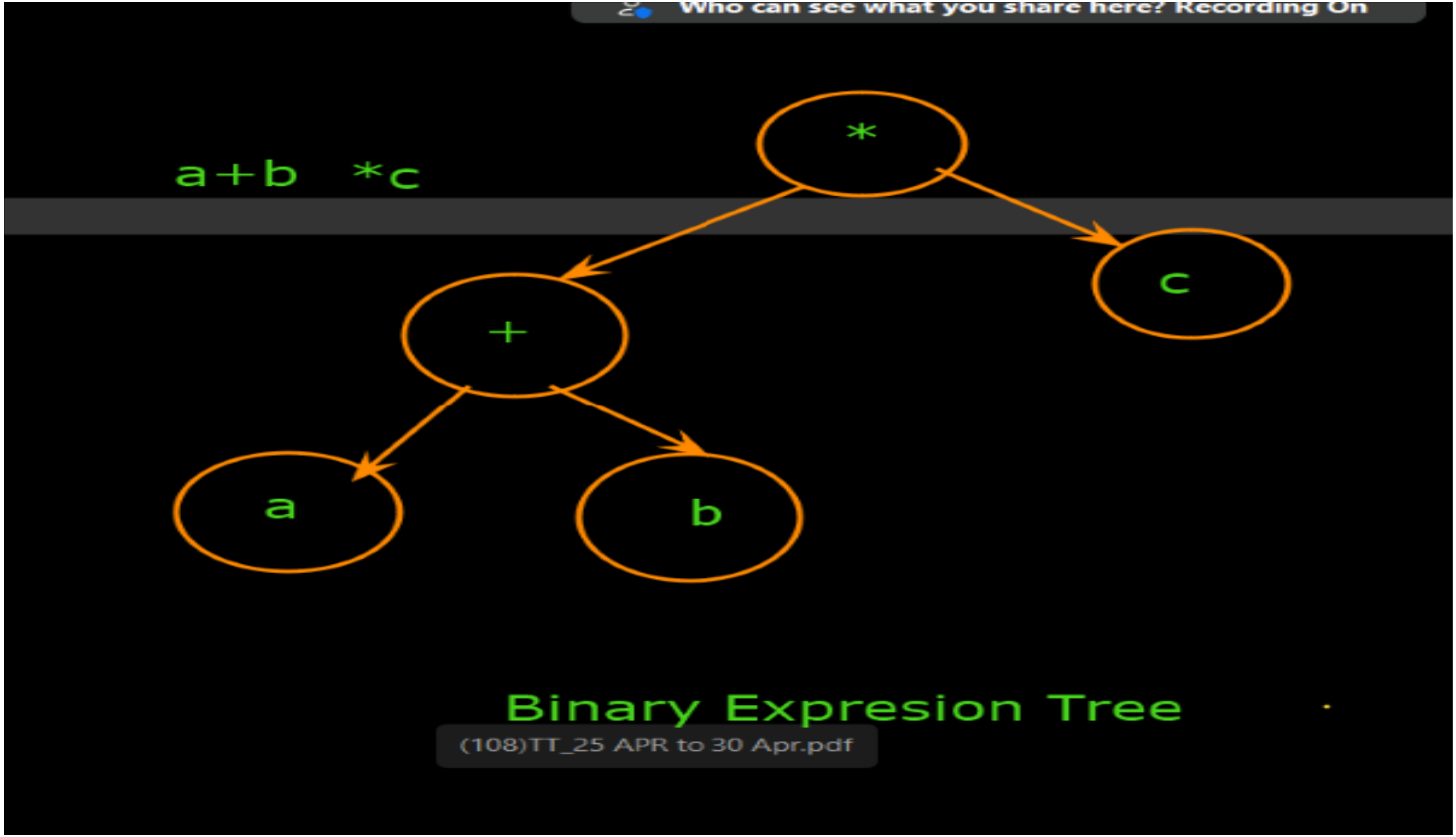
Steps:

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**

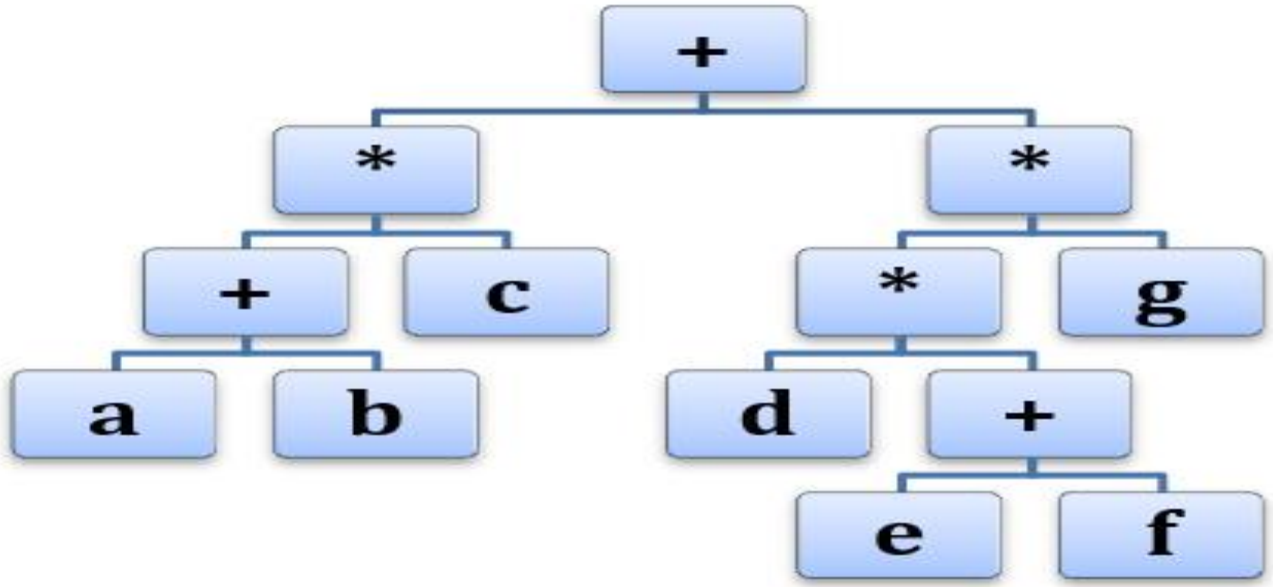
Postfix Evalaution:

3 10 5 + *

Character	Stack	Output
3	3	-
10	3, 10	-
5	3,10,5	-
+	3,15	5+10=15
*	-	3*15=45


Who can see what you share here? Recording On
a+b *c
*
+
c
a
b
Binary Expresion Tree
(108)TT_25 APR to 30 Apr.pdf

Construct a binary expression tree for the following statement :
(a + b * c) + ((d * e + f) * g)



Thanks