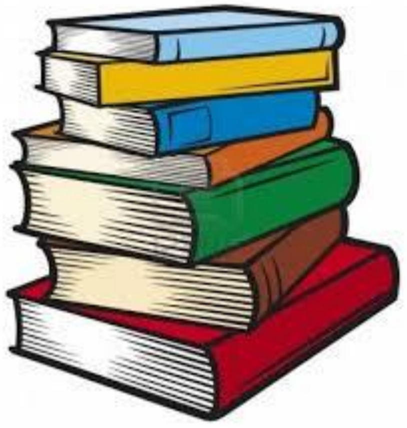
Algorithms & Data Structure

Kiran Waghmare

Examples of stack

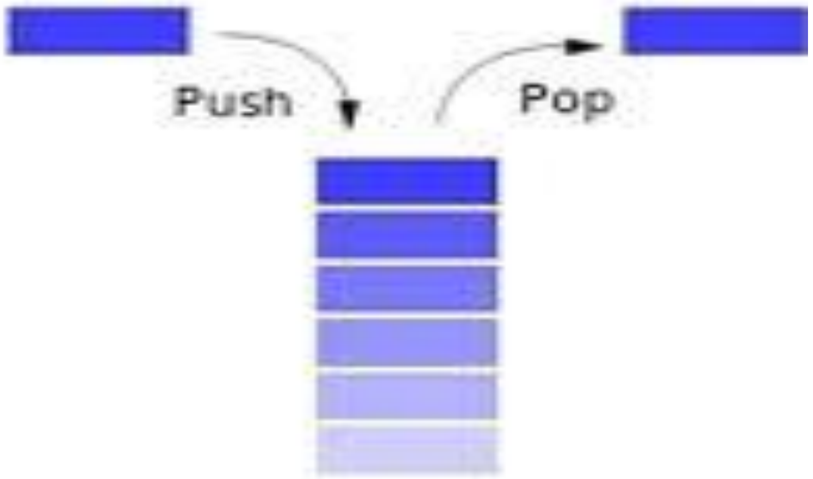
Stacks

Kiran Waghmare



Stack of books

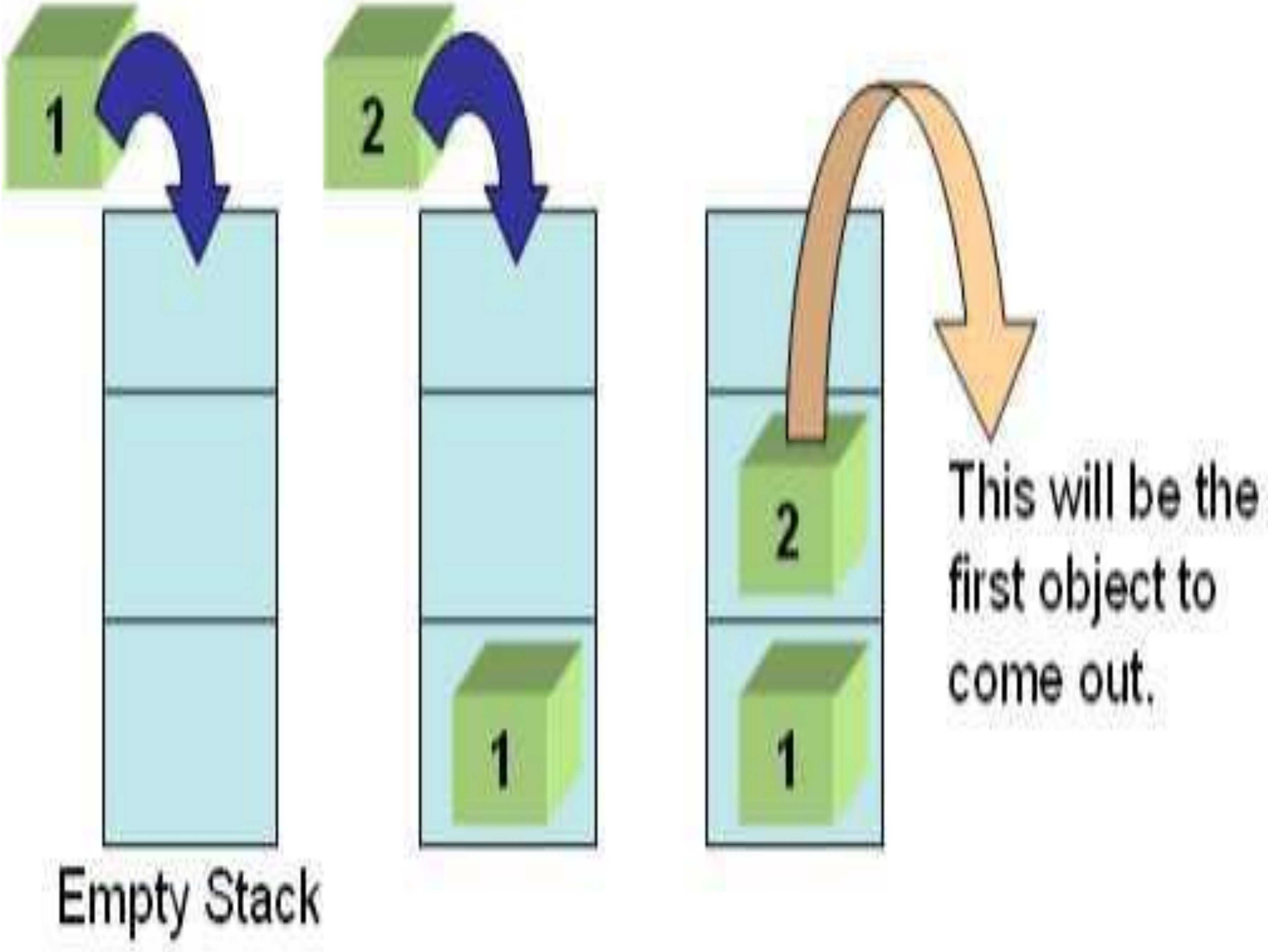
Stack of Coins

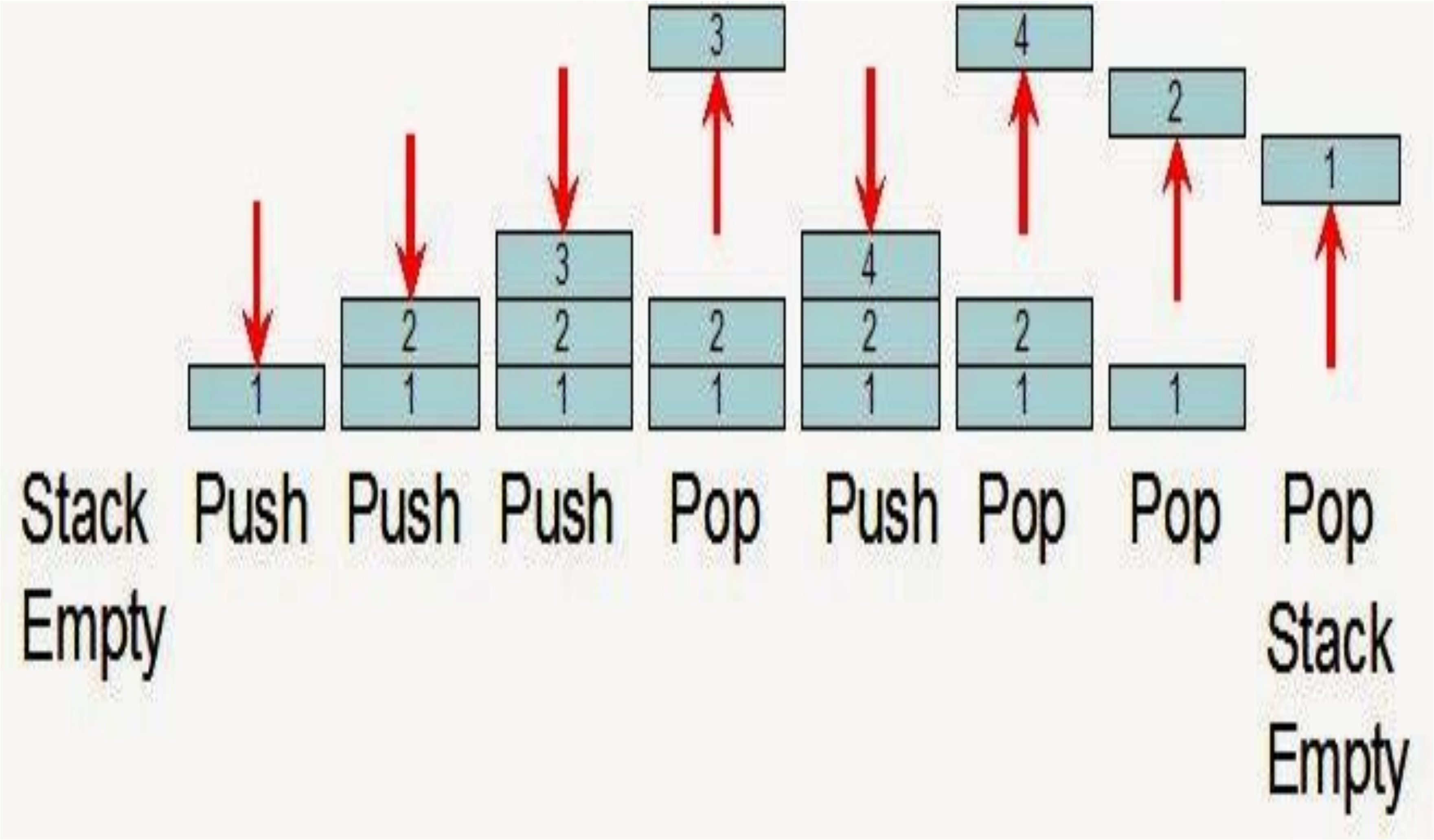


Memory stack

Standard Stack Operations

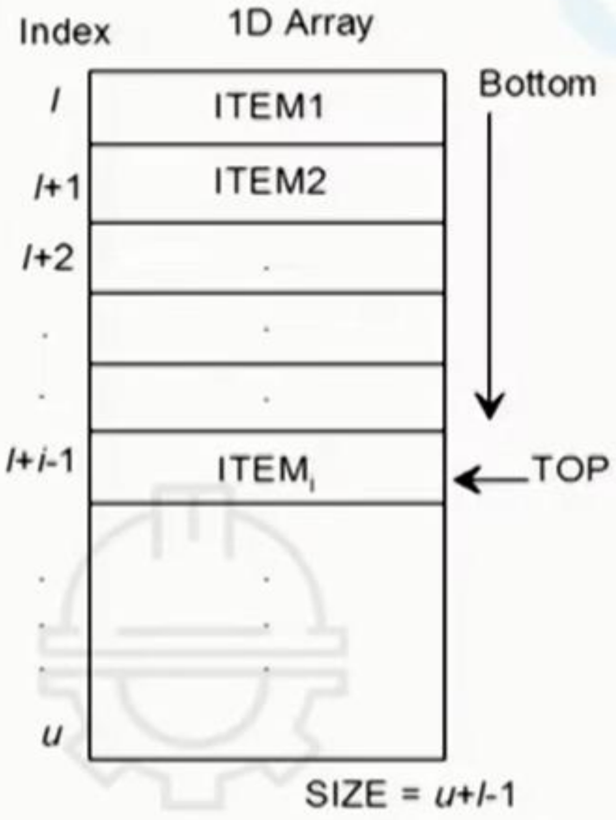
- **The following are some common operations implemented on the stack:**
- **push():**
  - When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():**
  - When we delete an element from the stack, the operation is known 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
1
2
1
Empty Stack
This will be the first object to come out.

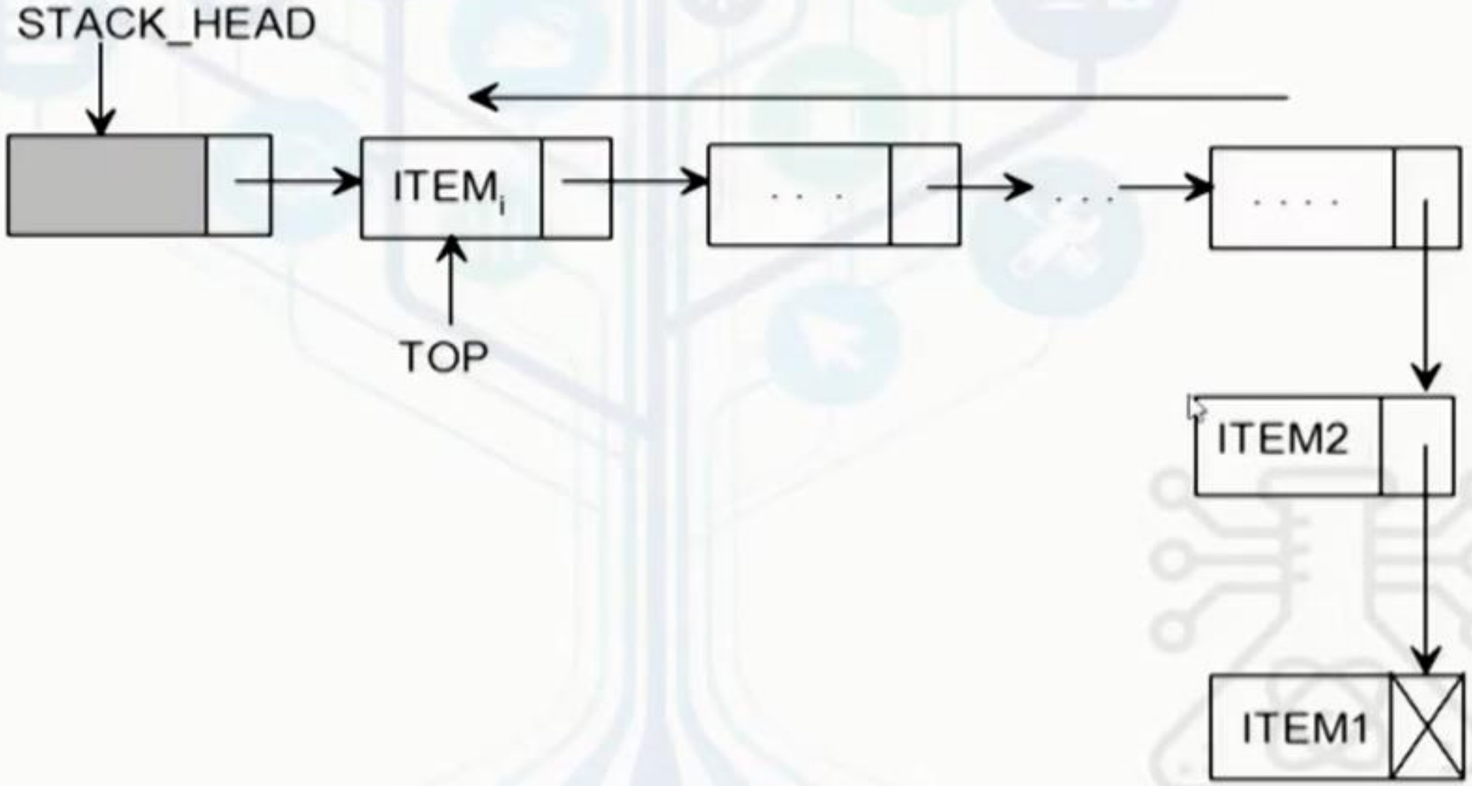

3
4
2
1
3
4
2
2
2
2
2
1
1
1
1
1
1
1
Stack Empty
Push
Push
Push
Pop
Push
Pop
Pop
Pop Stack Empty

Memory representations

Array representation

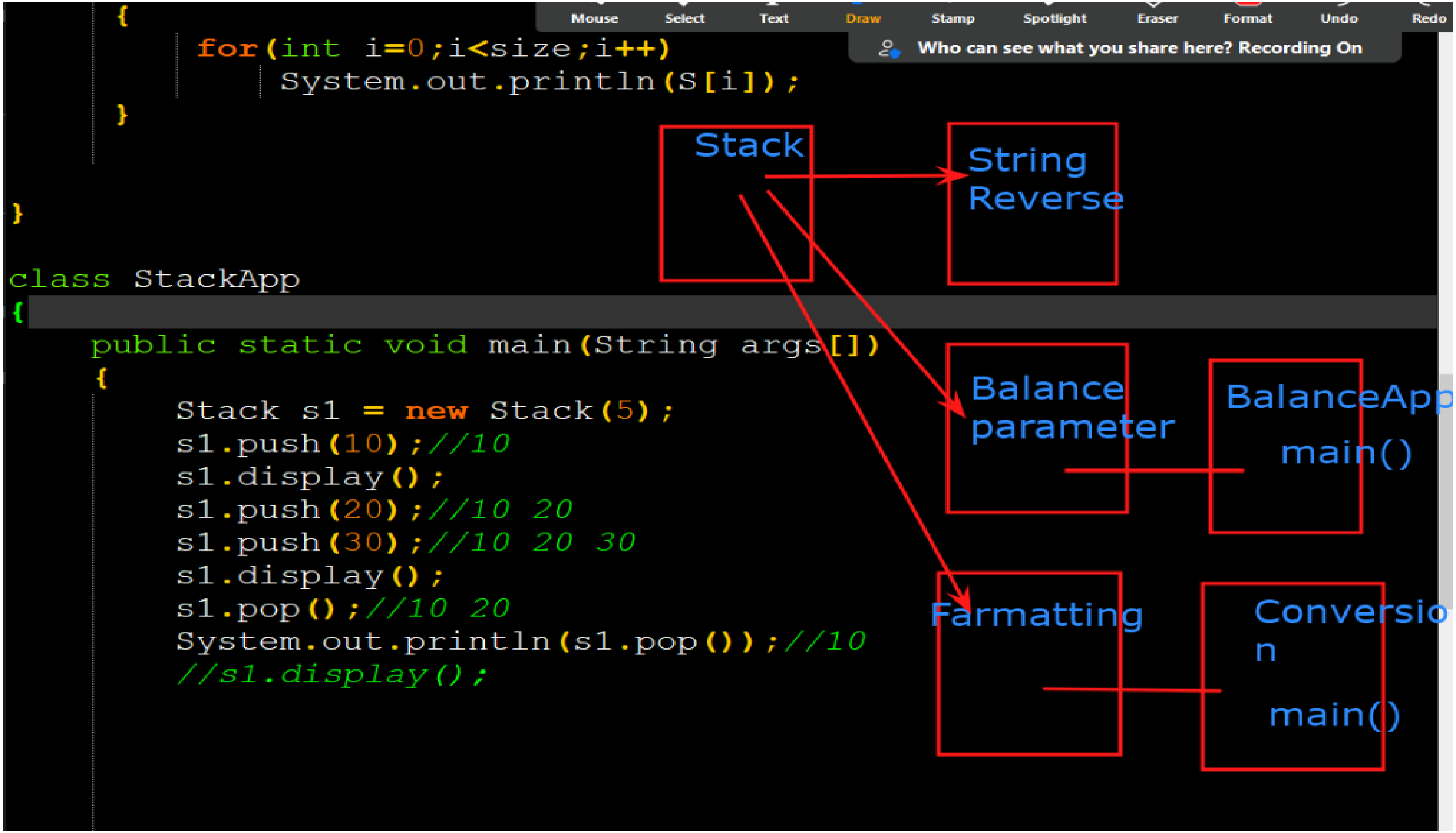


Linked list representation

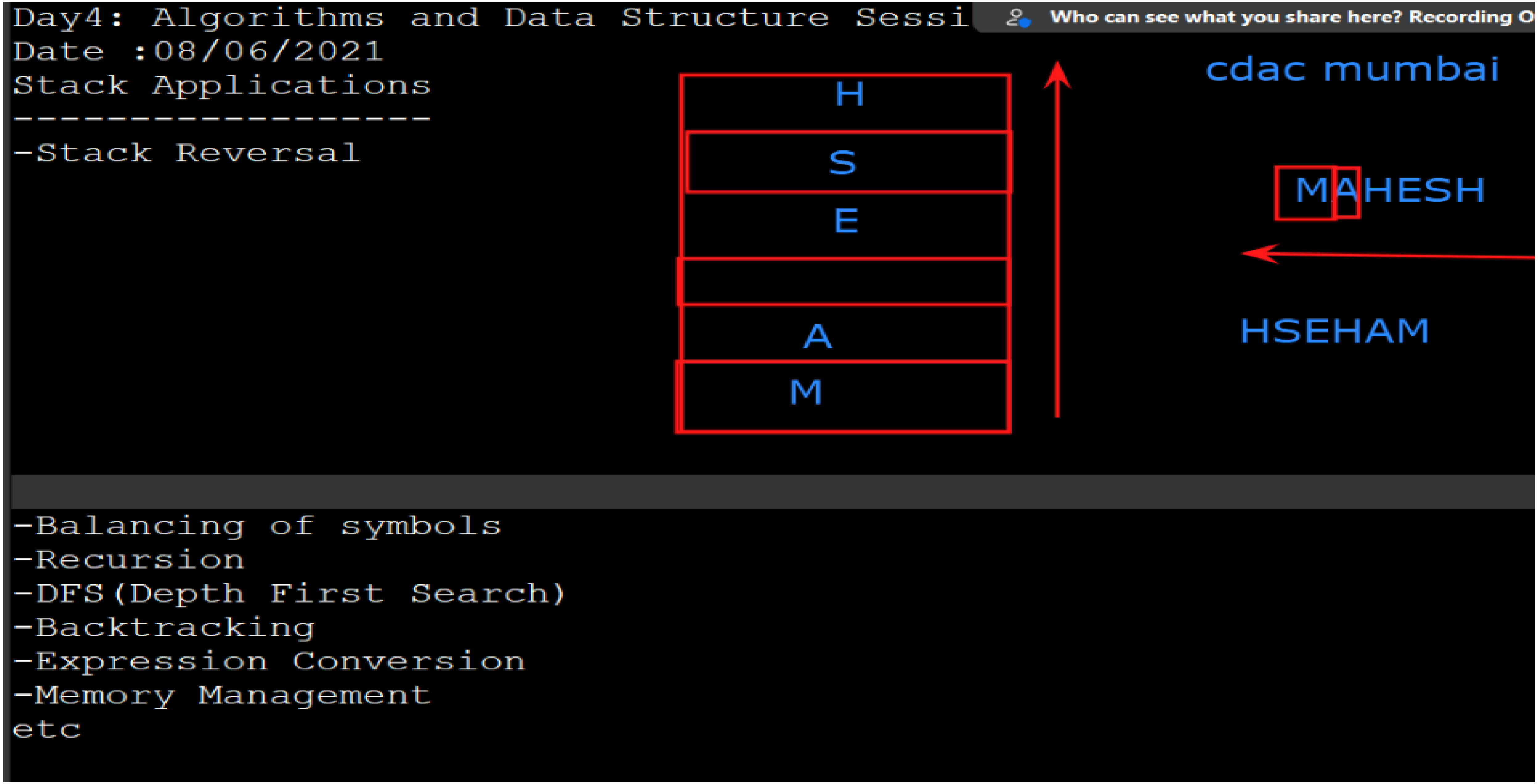


Applications of Stack

- The following are the applications of the stack:
- Balancing of symbols
- String reversal
- Expression conversion
- Recursion
- Backtracking



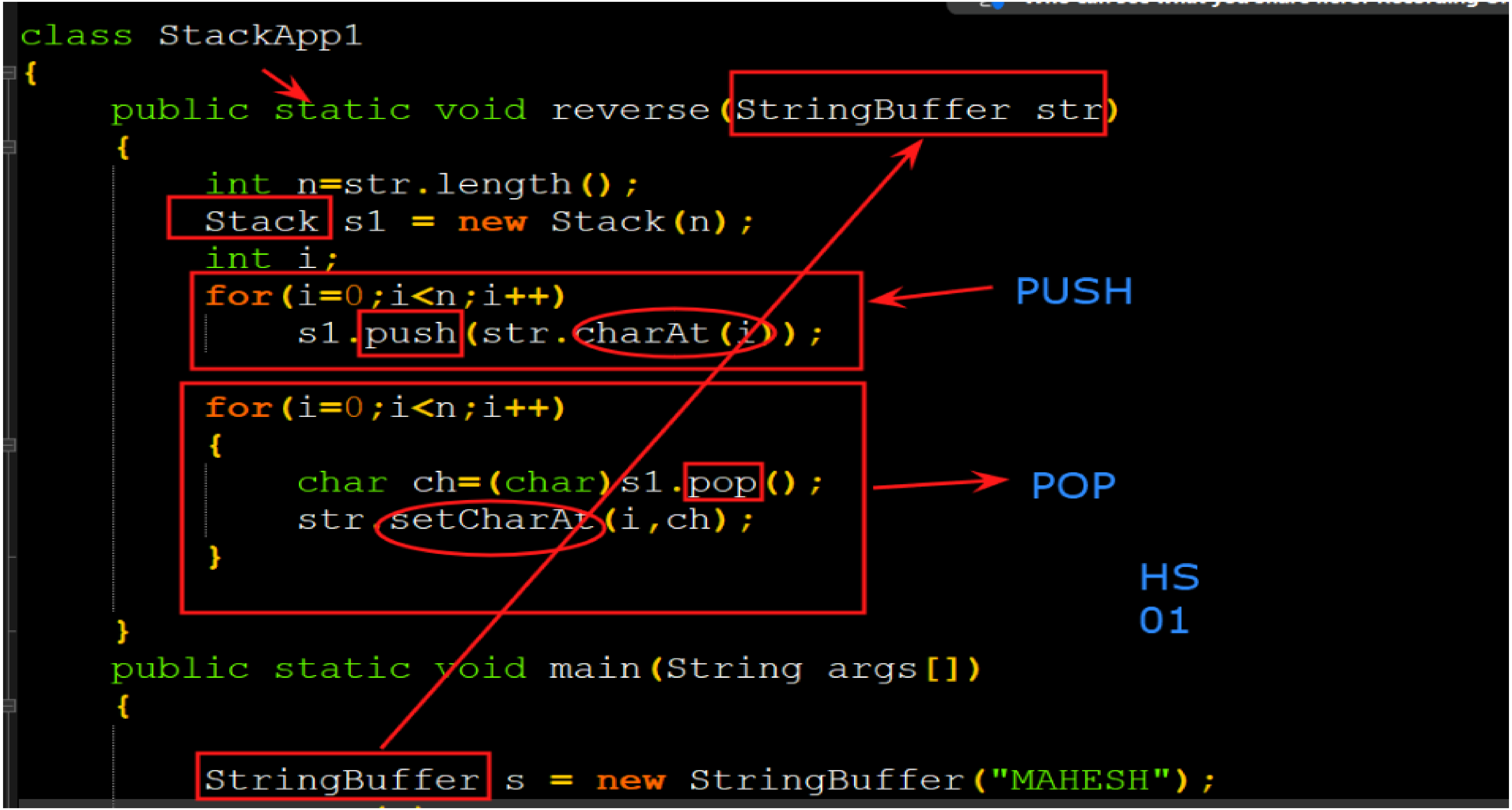
```
        {
            for(int i=0;i<size;i++)
                System.out.println(S[i]);
        }

}

class StackApp
{
    public static void main(String args[])
    {
        Stack s1 = new Stack(5);
        s1.push(10);//10
        s1.display();
        s1.push(20);//10 20
        s1.push(30);//10 20 30
        s1.display();
        s1.pop();//10 20
        System.out.println(s1.pop());//10
        //s1.display();
```



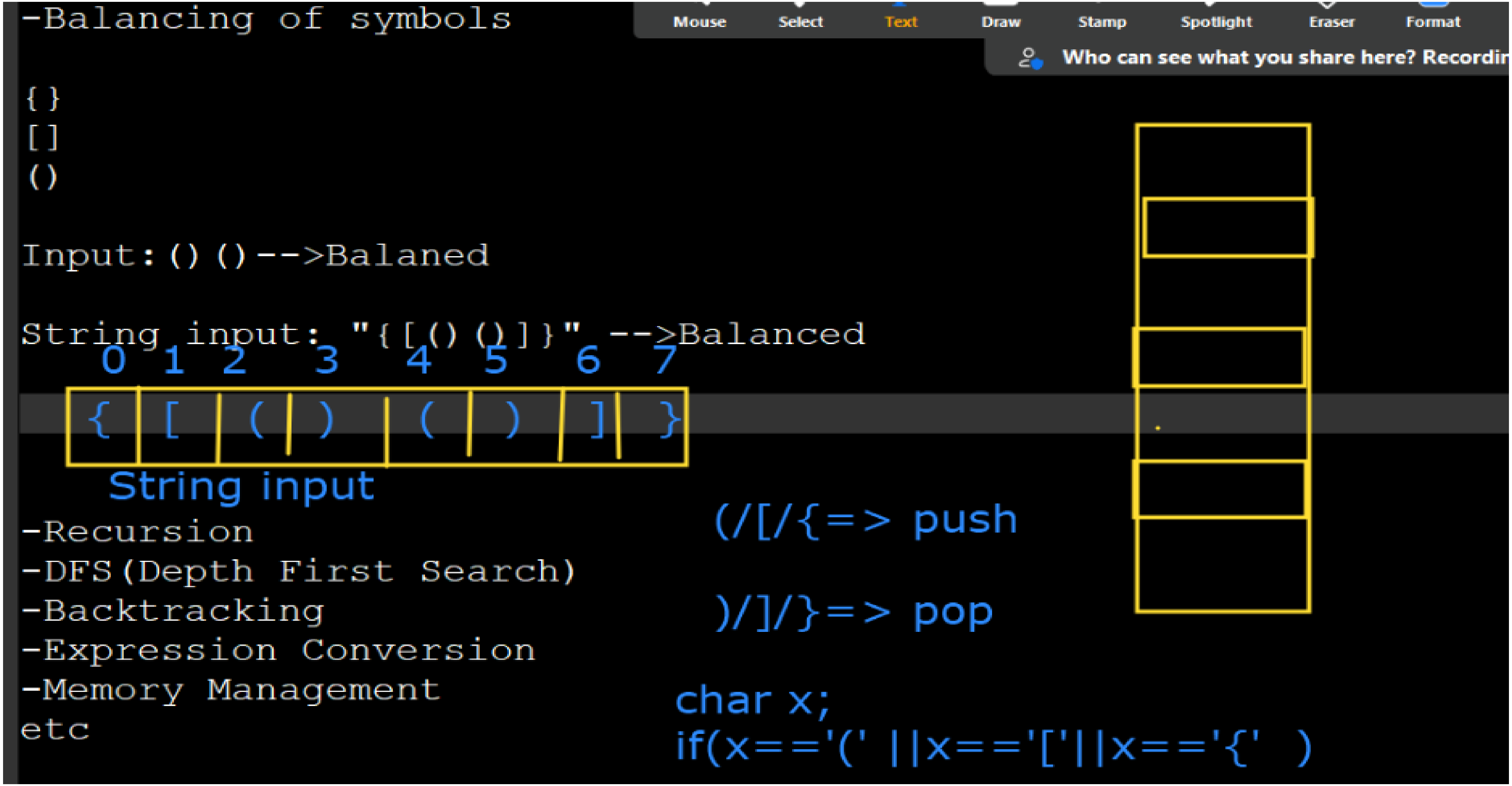
Day4: Algorithms and Data Structure Sessi
Date :08/06/2021
Stack Applications

-Stack Reversal

-Balancing of symbols
-Recursion
-DFS(Depth First Search)
-Backtracking
-Expression Conversion
-Memory Management
etc



```
class StackApp1
{
    public static void reverse(StringBuffer str)
    {
        int n=str.length();
        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("MAHESH");
```



```
-Balancing of symbols

{}
[]
()

Input:()()-->Balaned

String input: "{[()()]}" -->Balanced
```

0	1	2	3	4	5	6	7
{	[	(	)	(	)	]	}

String input

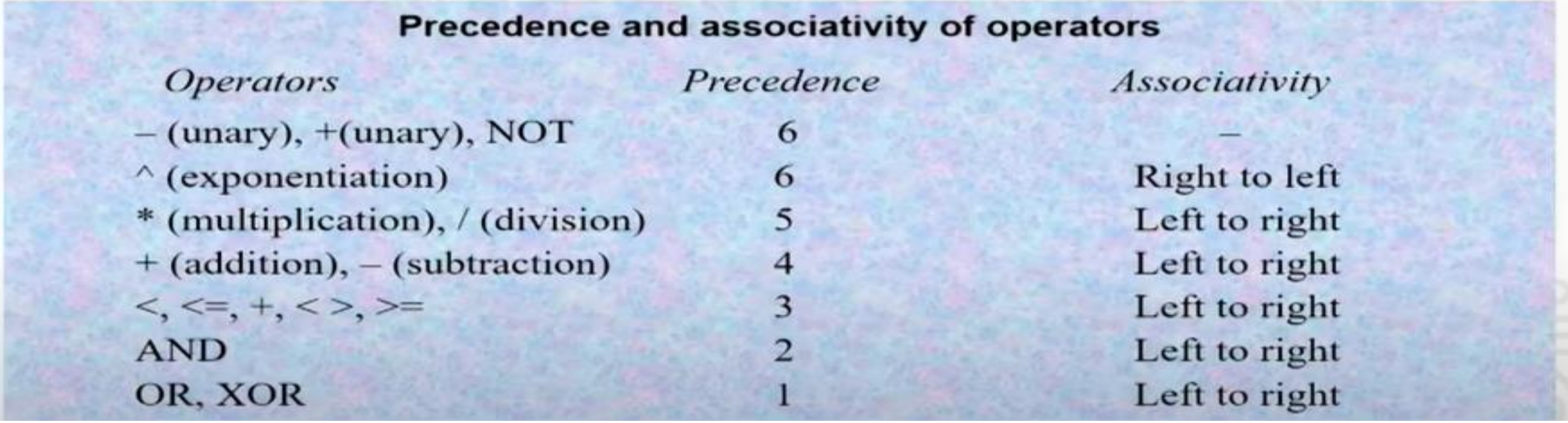
```
-Recursion
-DFS(Depth First Search)
-Backtracking
-Expression Conversion
-Memory Management
etc
```

(/[/{=> push

)/]/}=> pop

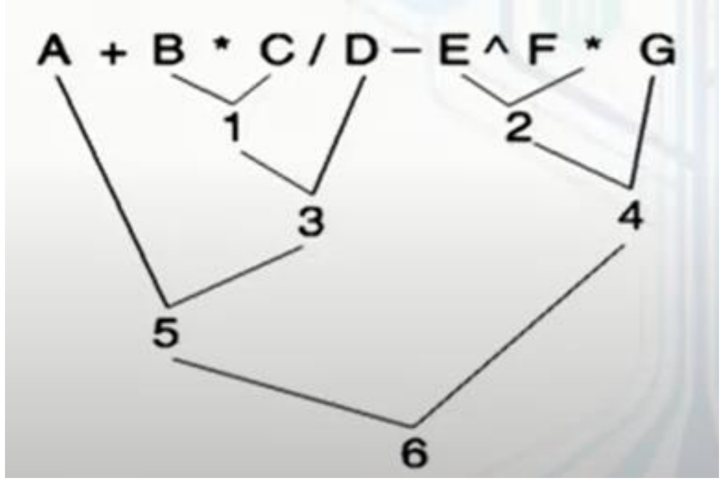
```
char x;
if(x=='(' ||x=='['||x=='{' )
```

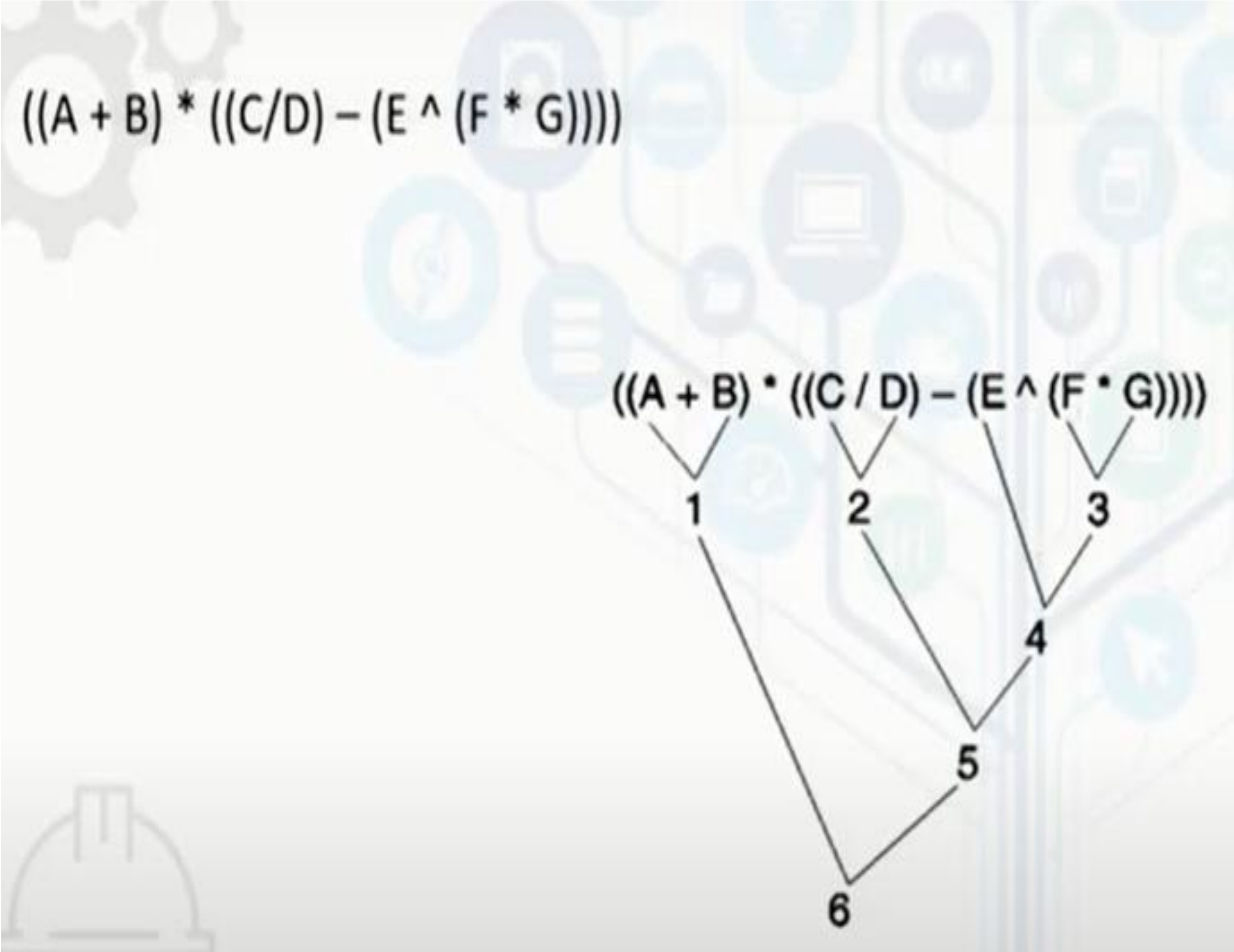
A + B * C / D – E ^ F * G



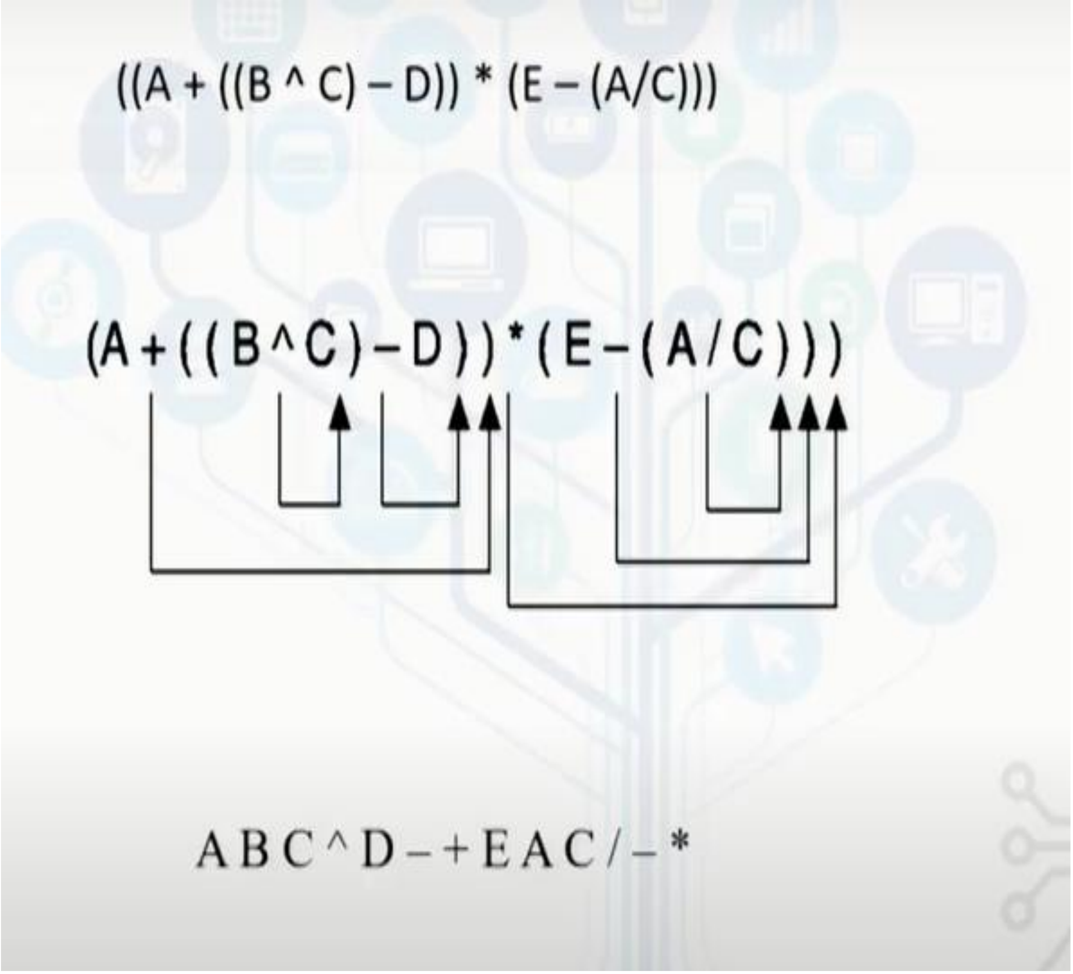
Precedence and associativity of operators

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



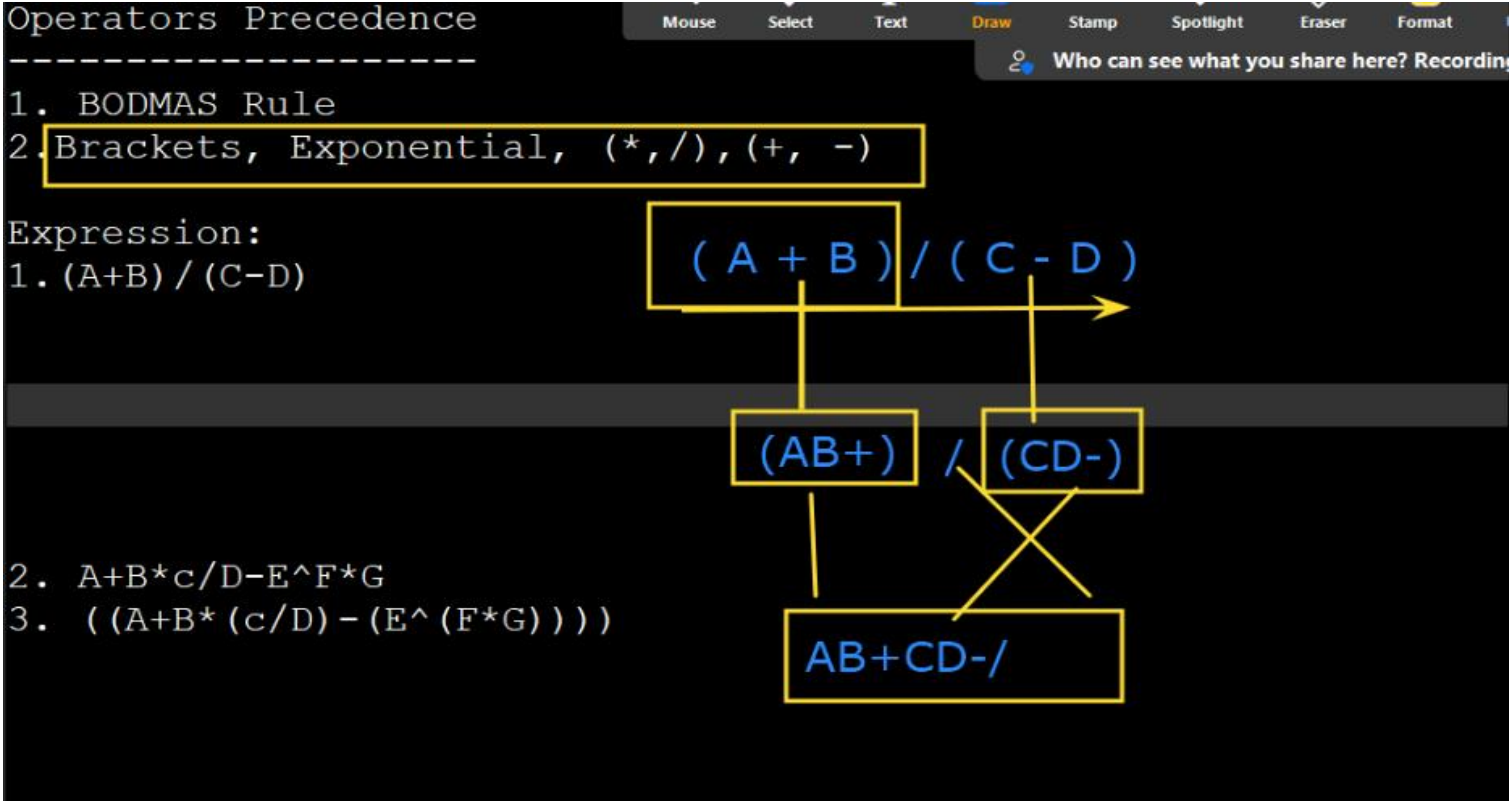


((A + B) * ((C/D) – (E ^ (F * G))))

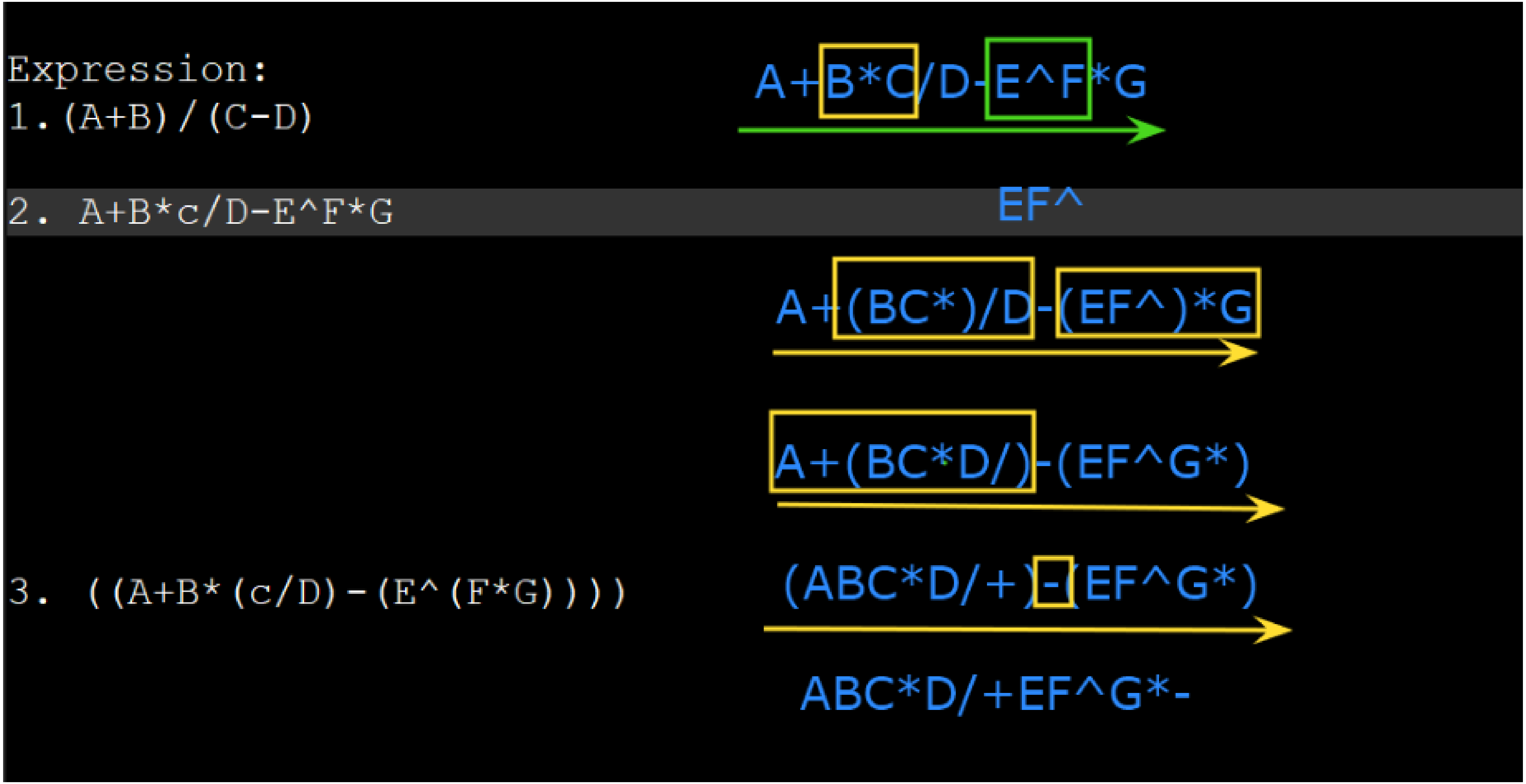


((A + ((B ^ C) – D)) * (E – (A/C)))

A B C ^ D – + E A C / – *



```
Operators Precedence
---------------------
1. BODMAS Rule
2. Brackets, Exponential, (*,/),(+, -)

Expression:
1.(A+B)/(C-D)

2. A+B*c/D-E^F*G
3. ((A+B*(c/D)-(E^(F*G))))
```



Expression:

1. (A+B)/(C-D)

2. A+B*c/D-E^F*G

3. ((A+B*(c/D)-(E^(F*G))))

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

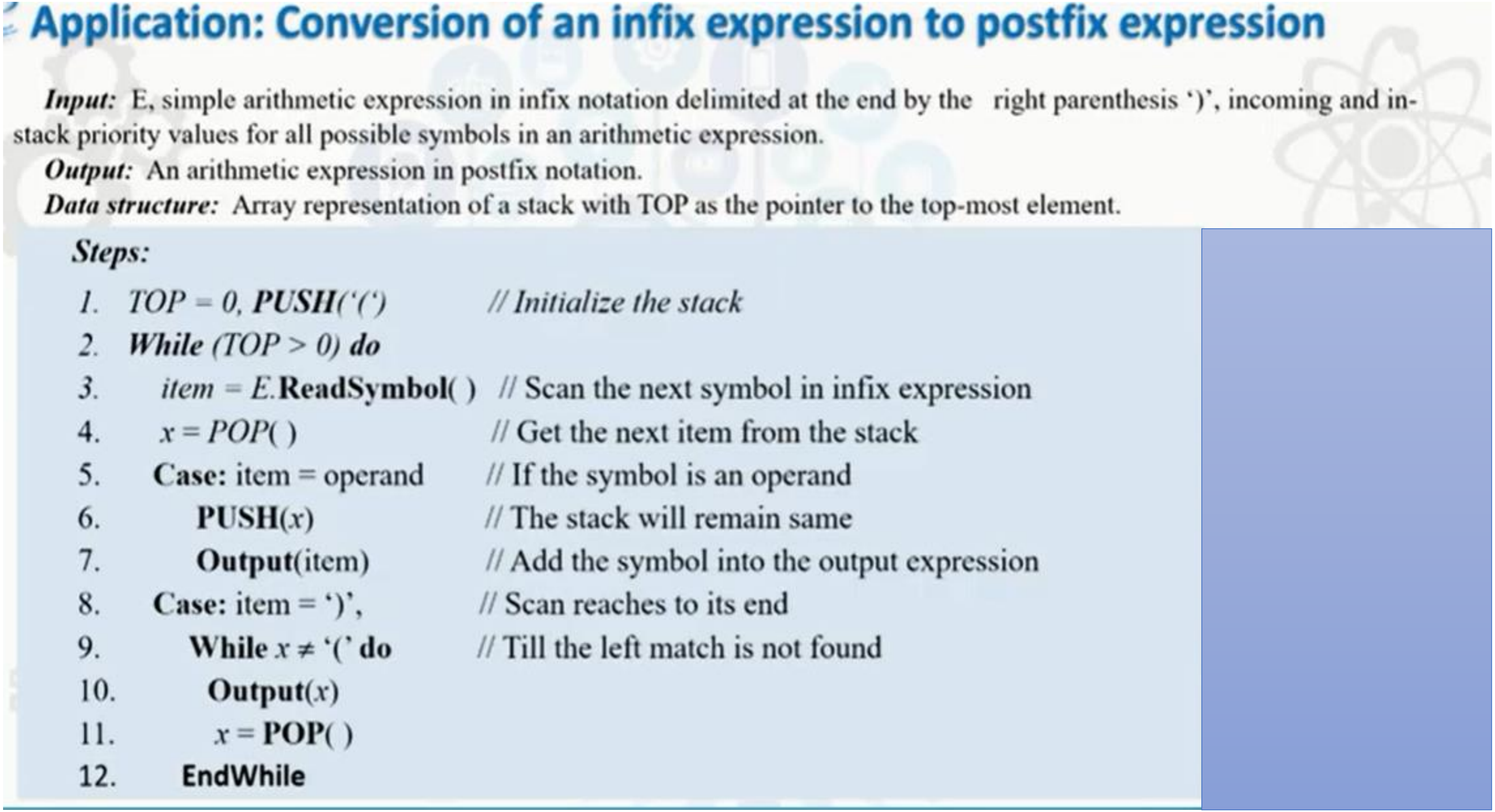
EF^

A+(BC*)/D-(EF^)*G

A+(BC*D/)-(EF^G*)

(ABC*D/+)-(EF^G*)

ABC*D/+EF^G*-



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.

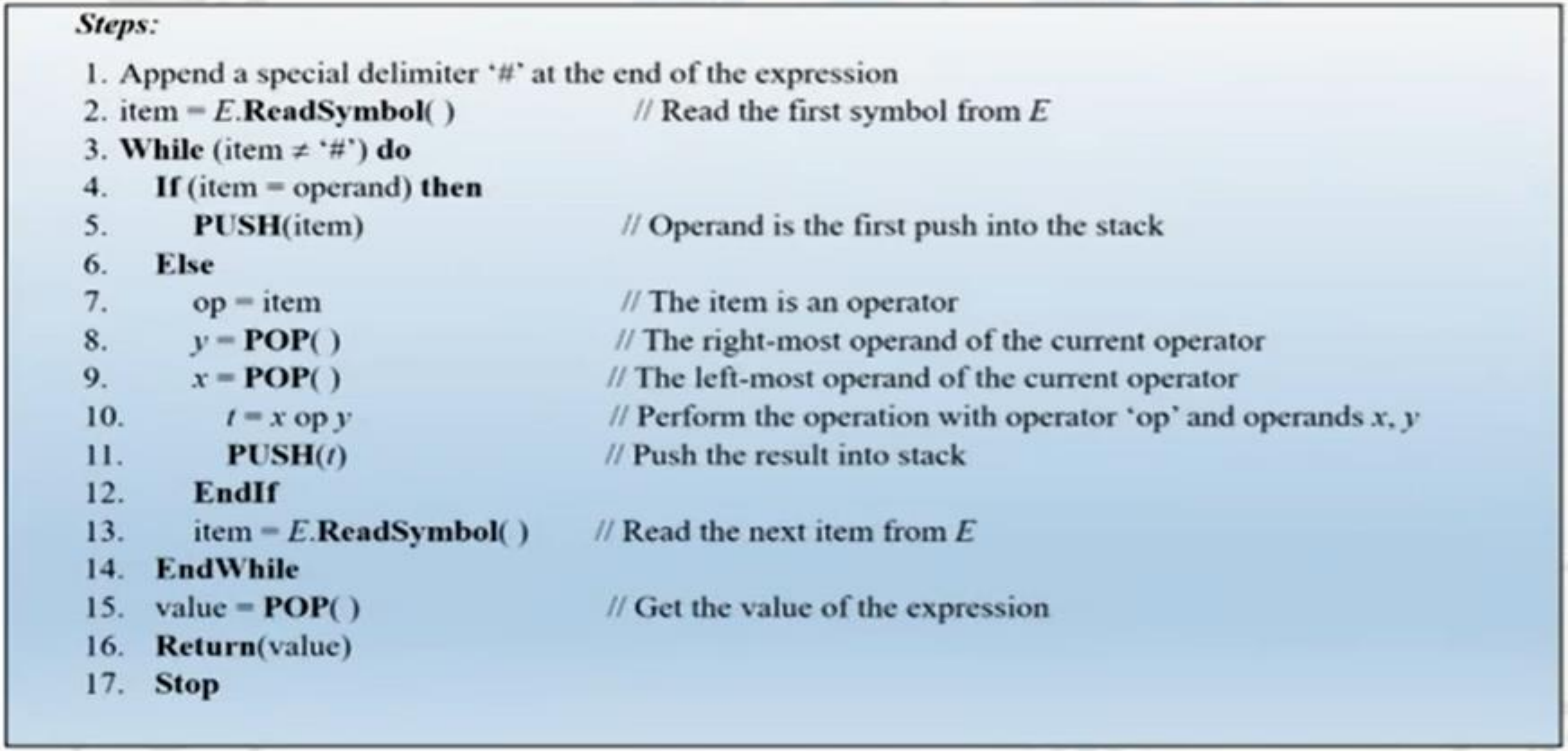
Output: An arithmetic expression in postfix notation.

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

Steps:

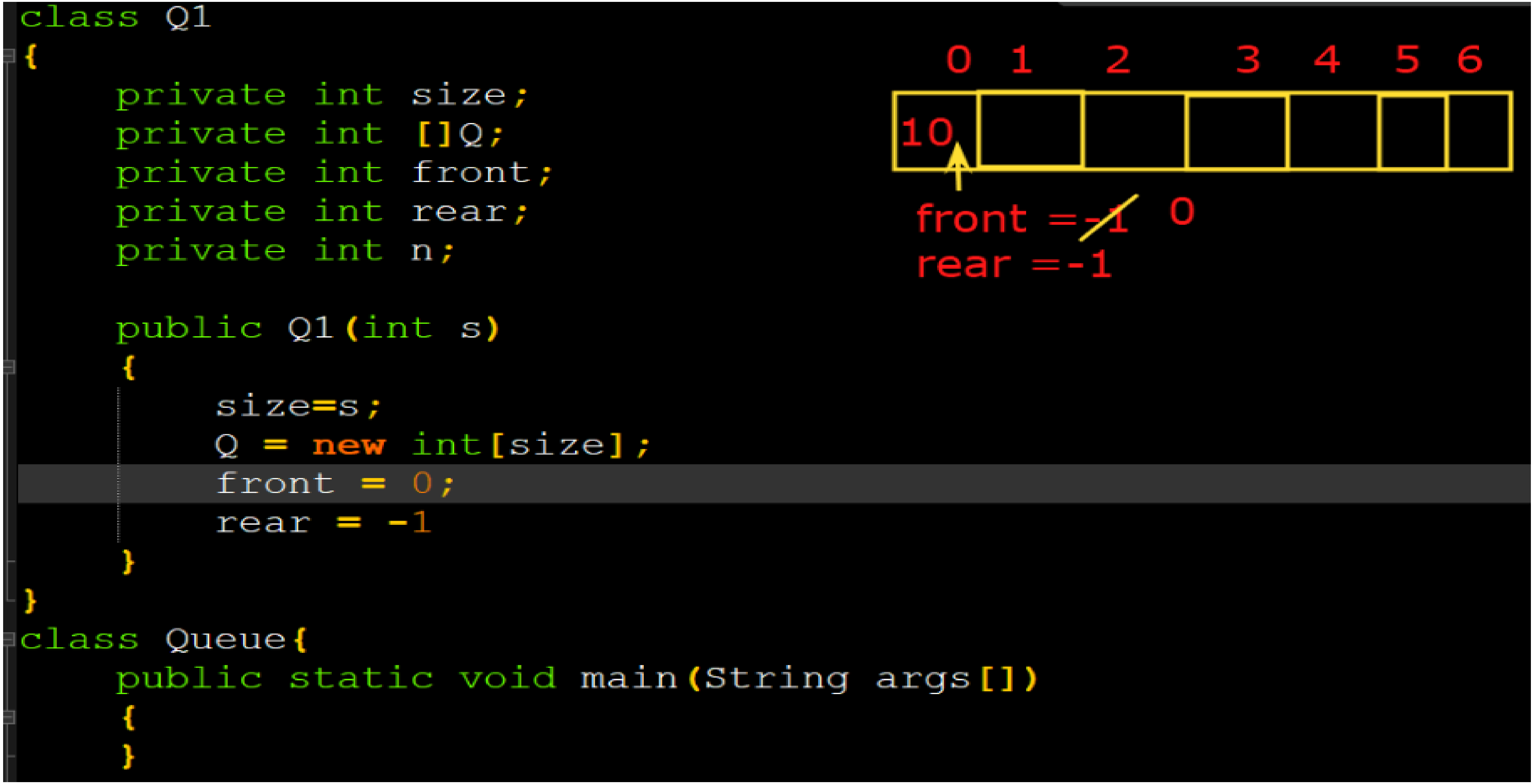
```
1.  TOP = 0, PUSH('(')            // Initialize the stack
2.  While (TOP > 0) do
3.      item = E.ReadSymbol( )    // Scan the next symbol in infix expression
4.      x = POP( )                // Get the next item from the stack
5.      Case: item = operand      // If the symbol is an operand
6.          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 $x \neq$ '(' do       // Till the left match is not found
10.             Output(x)
11.             x = POP( )
12.         EndWhile
```

Application: Evaluation of a postfix expression



Steps:

```
Append a special delimiter '#' at the end of the expression
item = E.ReadSymbol( )                 // Read the first symbol from E
While (item ≠ '#') do
    If (item = operand) then
        PUSH(item)                     // Operand is the first push into the stack
    Else
        op = item                      // The item is an operator
        y = POP( )                     // The right-most operand of the current operator
        x = POP( )                     // The left-most operand of the current operator
            t = x op y                 // Perform the operation with operator 'op' and operands x, y
            PUSH(t)                    // Push the result into stack
        EndIf
        item = E.ReadSymbol( )         // Read the next item from E
    EndWhile
    value = POP( )                     // Get the value of the expression
    Return(value)
    Stop
```



```
class Q1
{
    private int size;
    private int []Q;
    private int front;
    private int rear;
    private int n;

    public Q1(int s)
    {
        size=s;
        Q = new int[size];
        front = 0;
        rear = -1
    }
}
class Queue{
    public static void main(String args[])
    {
    }
```