



# Data Structure & Algorithms

*Sunbeam Infotech*

*Nilesh Ghule*



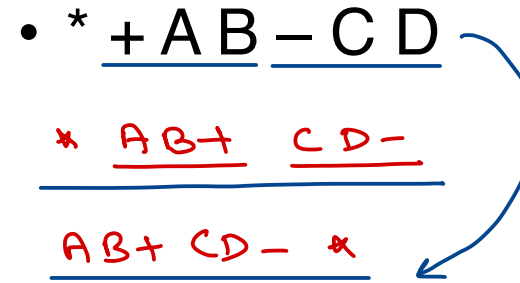
# Prefix to Postfix

- 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 Stack
- Create a string by concatenating the two operands and the operator after them.
- string = operand1 + operand2 + operator
- And push the resultant string back to Stack
- Repeat the above steps until end of Prefix expression.

•  $\ast \text{ } + \text{ } A \text{ } B \text{ } - \text{ } C \text{ } D$

$\ast \text{ } \underline{A \text{ } B +} \text{ } \underline{C \text{ } D -}$

$\underline{A \text{ } B + \text{ } C \text{ } D - \text{ } \ast}$

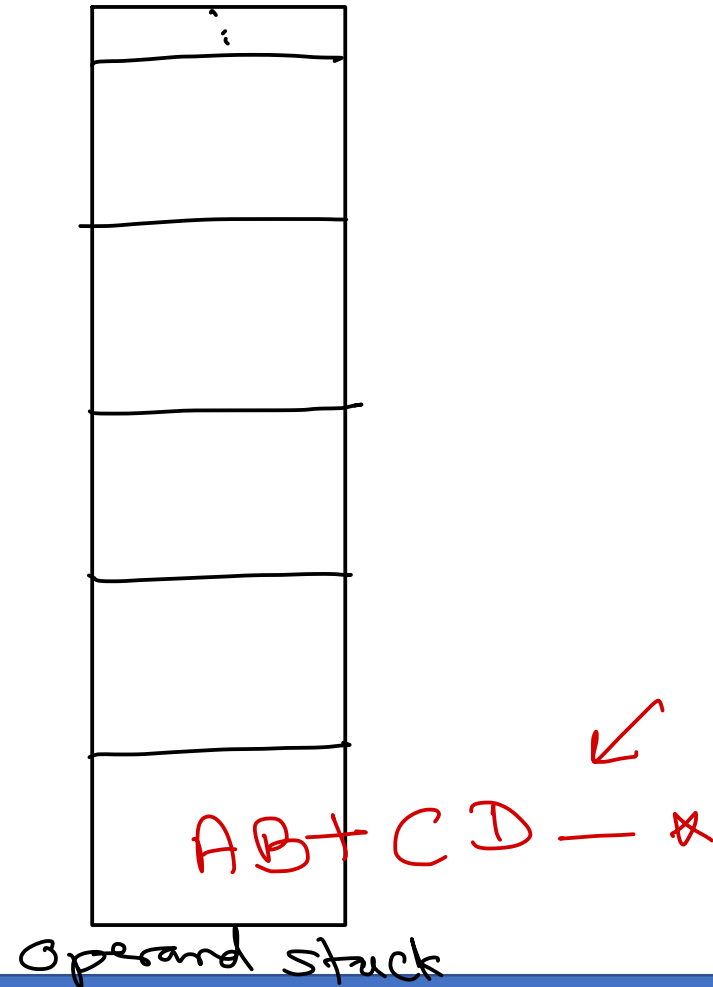




# Prefix to Postfix

- 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 Stack
- Create a string by concatenating the two operands and the operator after them.
- string = operand1 + operand2 + operator
- And push the resultant string back to Stack
- Repeat the above steps until end of Prefix expression.

• \* + A B – C D



# Parenthesis Balancing

•  $5 + ([9 - 4] * (8 - \{6 / 2\}))$



open	(	[	{	<
closing	)	]	}	>
	↑	↑	↑	↑
	0	1	2	3




# applications

- ① process stack
  - ↳ program in execution
  - function call/recursion
    - ↳ FAR/SF created on stack.
    - ↳ local vars, args, return addr.

- ② JVM memory area
  - ↳ stack for method execution.

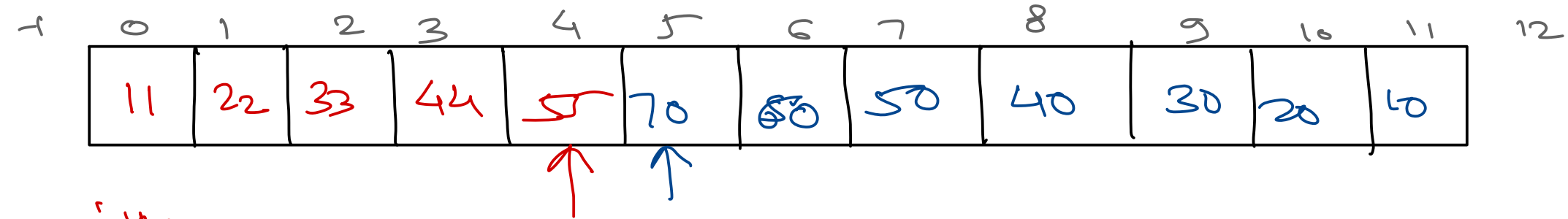
- ③ Calculators/Compilers - lang syntax
  - ✓ infix, postfix, prefix.
  - ✓ parenthesis balancing

- ④ undo/redo.

- ⑤ graph → dfs  
tree → non recursive  
preorder, inorder, ...



implement two stacks in single array.



init:  
 $top1 = -1;$

empty:  
 $top1 == -1;$

full:  
 $top1 + 1 == top2$

init2:  
 $top2 = size;$

empty2:  
 $top2 == size;$

push2:  
 $top2--;$   
 $arr[top2] = val;$

pop2:  
 $top2++;$

peek2:  
 $\text{return } arr[top2];$

push1:  
 $top1++;$   
 $arr[top1] = val;$

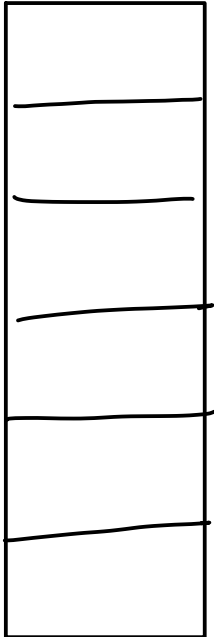
pop1:  
 $top1--;$

peek1:  $\text{return } arr[top1];$



# Stack / Queue – Competitive Programming

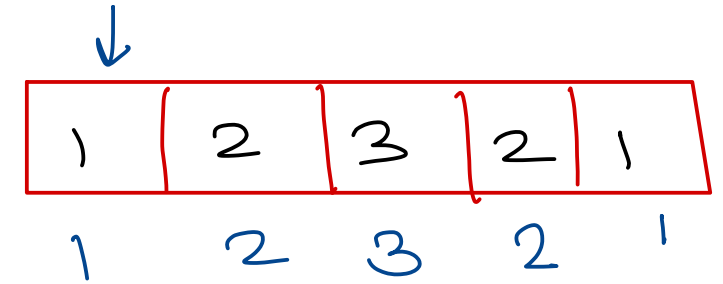
- Reverse array, string or linked list.



```
for (i = 0; i < size; i++)  
    s.push(arr[i]);
```

```
i = 0;  
while (!s.isEmpty()) {  
    arr[i] = s.pop();  
    i++;  
}
```

3



```
for (i = 0; i < size; i++)  
    s.push(arr[i]);
```

```
i = 0;  
while (!s.isEmpty()) {  
    if (arr[i] != s.pop())  
        return false;  
    i++;  
}
```

3

```
return true;
```



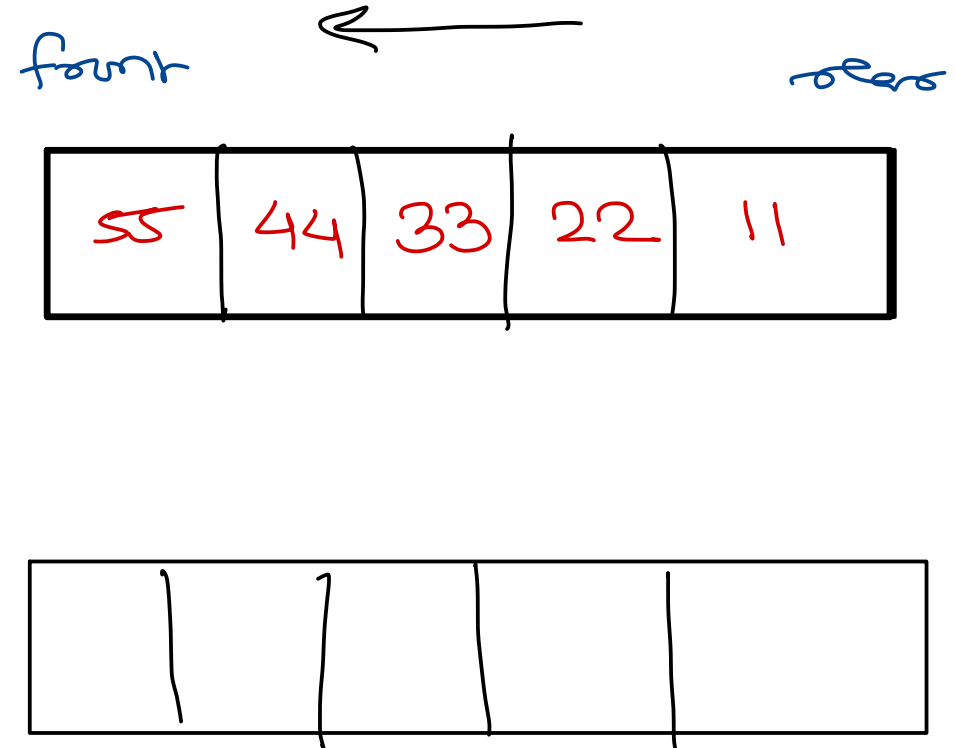
# Stack / Queue – Competitive Programming

- Create stack using queue.

push:  $\rightarrow T \propto 2n \rightarrow O(n)$

while (!que.is Empty ())  
    tempq.offer(que.poll());  
que.offer(val);  
while (!tempq.is Empty ())  
    que.offer(tempq.poll());

pop:  $\rightarrow O(1)$   
return que.poll();





# Stack / Queue – Competitive Programming

- Create ~~stack~~ using ~~queue~~.  
queue      stack.



# Rat in a Maze - Backtracking

```
class Cell {  
    int r, c;  
}
```


Stack<Cell> s;

- ① push start cell on stack.  
mark start cell as visited
- ② pop cell from stack. → cur cell
- ③ if cell is dest cell, return.
- ④ put all non-obstacle, non-visited neighbors on stack & mark them as visited.
- ⑤ repeat steps 2-4 until stack is empty.

boolean[][] visited; → all ele = false initialed.


	0	1	2	3	4	5	6	7
0	✓	✓				✓	✓	
1		✓	✓				✓	
2		✓	✓				✓	✓
3		✓	✓					✓
4			✓	✓	✓	✓	✓	✓
5				✓			✓	
6								
7								D

4,7  
4,4

	0	1	2	3	4	5	6	7
0	S		1	1	1			1
1	1			1	1	1		1
2	1			1	1	1		
3	1			<del>3,3</del>	1	1	1	
4		1	4,2	4,3	4,4			
5		1	1	5,3	1			1
6	1		1	1		1		1
7				1				D



*Thank you!*

Nilesh Ghule <nilesh@sunbeaminfo.com>

