Stacks 1

Good Morning

Todays content

01. Stack basics
02. Implementation of Stack
    - → Using Arrays
    - → Using Linkedlist
03. Balanced brackets
04. Evaluate expression
05. Nearest smaller element on left.

Stack → Linear data structure that stores data/ information in a sequence, from bottom to top.

→ Data can only be accessed from top.

↓

Last In First Out

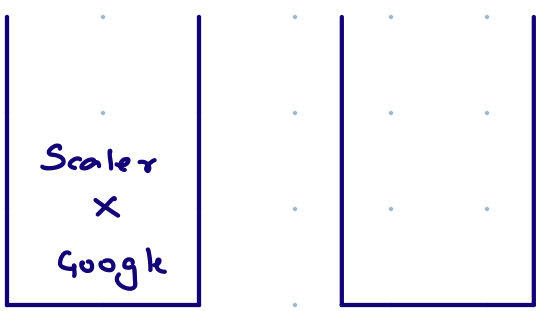
Real world Examples

01. Piles of plate
03 Stack of chair
03. Book stack
04. Boxes in tiffin box
05. Recursive stack

→ All function goes inside stack place at the top & once the top most function gets executed, then only we move down.

06. Browser History → Google → X → Scaler → YT

←
←
→
→



Stack

```
Stack <Datatype> st = new Stack<>();
```

Operations

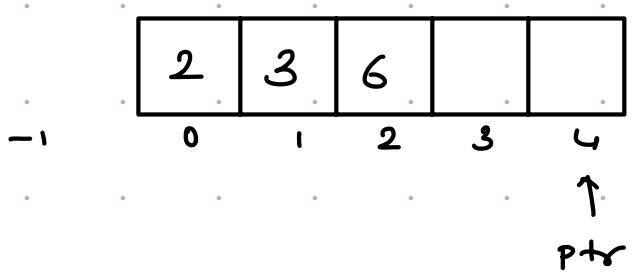
01. Add (x) ⟶ st.push(x)
02. remove ( ) ⟶ st.pop( )
03. Access topmost element ⟶ st.peek( )
04. size ⟶ st.size( )

Tc : O(1)

* Implementation of Stack

01. Using Arrays

push (2)
push (3)
push (10)



pop( ) → return A[2] ⇒ 10
reduce ptr by 1

push (6)

ptr: ~~4~~ ~~3~~ ~~2~~ 2

peek( ) → return A[2] ⇒ 6

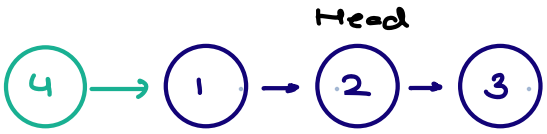
pop( ) → return A[2] ⇒ 6
reduce ptr by 1

```
class Stack {

    int [ ] A = new int [5];
    int ptr = -1;

    void push ( x )
    {
        if ( ptr == A.length - 1 )
            return ;    // overflow
        ptr = ptr + 1;
        A [ptr] = x ;
    }

    ---- pop
    {
        if ( ptr == -1 )
            return -1 ; // underflow
        int val = A [ ptr ]
        ptr = ptr - 1 ;
        return val ;
    }

    ---- peek ( )
    {
        if ( ptr == -1 )
            return -1 ;
        int val = A [ ptr ]
        return val ;
    }

    int size ( )
    {
        return ptr + 1 ;
    }
}
```

* Implementation using Array has a disadvantage that we have to fix size of stack initially

* Implementation using LinkedList

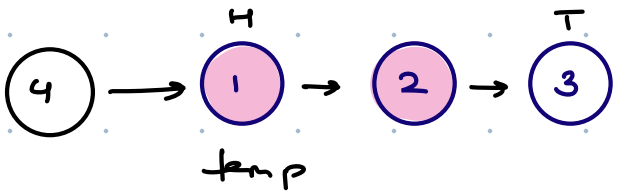
Last In First Out



push(4)

Insert at head → TC : O(1)

Delete at head → TC : O(1)



push(3)

push(2)

push(1)

1 ← pop()

2 ← pop()

push(4)

Insert at tail = O(1)

Delete at tail = O(n)

} Valid Stack but with bad TC

Balanced Paranthesis

Q Check whether the given sequence of paranthesis is valid / balanced

→ For last closing brackets → the last opening bracket should match

→ For all opening brackets → there should be a closing bracket

Eg 1 = ( ) [( )] { } → True

Eg 2 = (( ) } → false

Eg 3 = (}) { → false

Eg 4 = ({) } → false

Eg 5 = ([{] }) → false

Eg 6 = { [[ ] { }] } → true

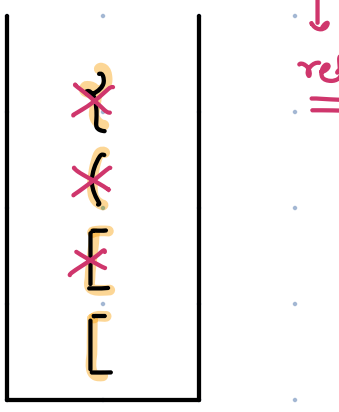
A	(a + b) * c → True
B	(a + b)) * c → false
C	(a + b) (c → false
D	(a + b) c) → False

Idea → For an expression to be valid, its sub expression should also be valid

↳ all closing brackets, match them with opening brackets of same type & then remove them.

> Note ⇒ Travel on the previously travelled element

str = " [[( ) { }]] }

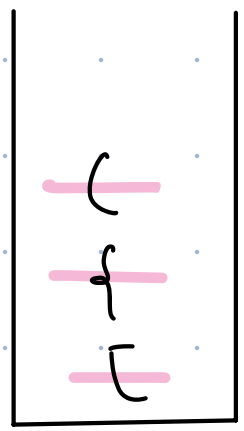



```
Stack < character > st ;

for ( i=0; i<N; i++ )
    char ch = str.charAt(i)
    if ( ch == '(' || ch == '[' || ch == '{' ) st.push(ch)
    else if ( ch == ')' )
```

TC: O(n)
SC: O(n)

```
        if (st.size() != 0 && st.peek() == '(') st.pop();
        else return false;
    }
    else if (ch == ']')
    {
        if (st.size() != 0 && st.peek() == '[') st.pop();
        else return false;
    }
    else if (ch == '}')
    {
        if (st.size() != 0 && st.peek() == '{') st.pop();
        else return false;
    }
}
return st.size() == 0
```

str = " [{ } ( )] "



2 * 3 * 4

↓

23 * * 4 ⇒ 23 * 4 *

Evaluate Expressions

Infix expression

Operand 1 Operator Operand 2

Postfix Expression

Operand 1 Operand 2 Operator

01. a + b

a b +

02. a * b

a b *

03. a + b * c

b c *

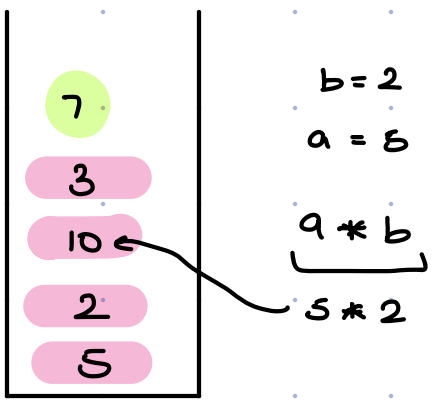
a b c * +

04. (a+b) * c

a b +

a b + c *

05. Evaluate 5 2 * 3 −



b = 2

a = 5

a * b

5 * 2

b = 3

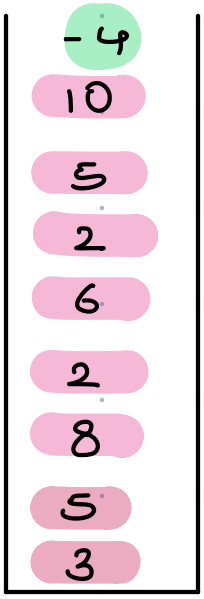
a = 10

a − b

10 − 3

7

06. Evaluate 3 5 + 2 − 2 5 * −



b = 5
a = 3
a+b
3+5

b = 2
a = 8
a−b
8−2

b = 5
a = 2
2 * 5

b = 10
a = 6
a−b
6 − 10 = −4

Code

```
Stack<Character> st = new Stack<>();

for (i=0; i<n; i++){
    char ch = str.charAt(i);
    if (ch == operator){
        b = get the first popped value
        a = get the second popped value:
        st.push(a op b)
    }
    else st.push(ch);
}
return st.pop();
```

* Given an integer A[]. Find the nearest smaller element idx on left.

	0	1	2	3	4	5
A[] = {	4	5	10	3	12	6 }
Ans[] = {	-1	0	1	-1	3	3 }

	0	1	2	3	4	5	6	7
A[] = {	8	2	4	7	9	5	3	4 }
Ans[] = {	-1	-1	1	2	3	2	1	6 }

A[] = { 8 _ _ _ 5 x y z }

For this x, y, z can index 0 ever be the answer ?