Stacks 1

Contest

→ HashMap

→ String

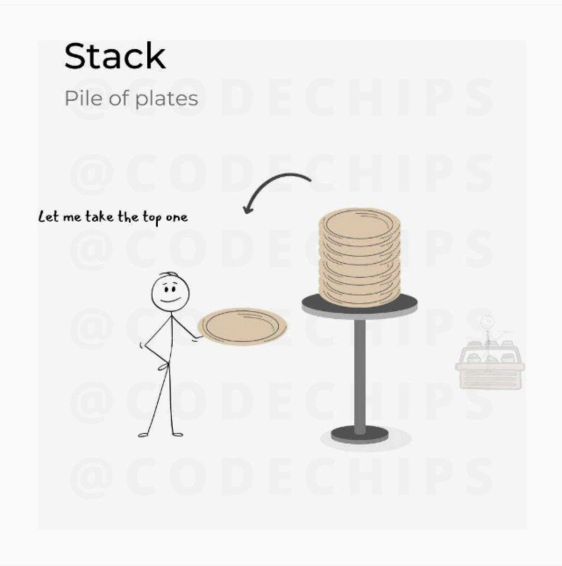
→ LL

Good Evening

Content for Today

01. Stack basics
02. Implementation of Stack
    - → Using Arrays
    - → Using Linkedlist
03. Balanced brackets
04. Double trouble
05. Postfix Evaluation

Stack → Pile of objects



Real life examples

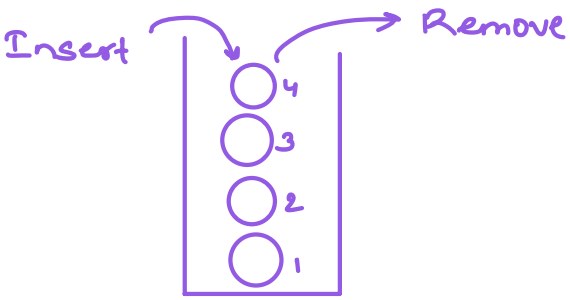
- → Loaves of bread
- → Piles of plate
- → Bricks
- → Box of balls

Stack → LIFO

Last In First Out

FILO

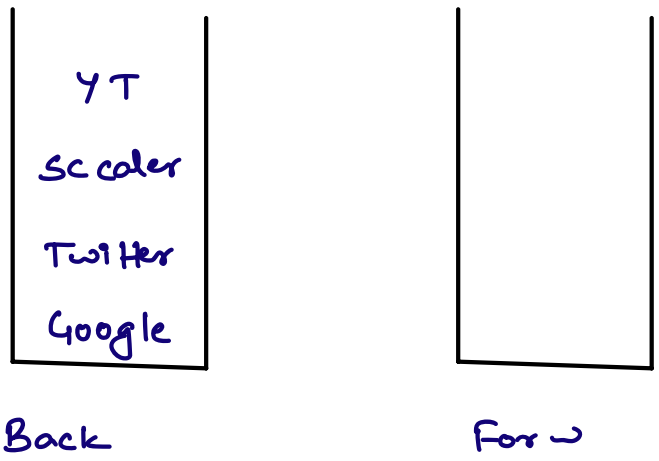
First In last out



* Recursion → All the calls are getting stacked one above the other in recursive stack.

* Browser history

Google → Twitter → Scaler → YT



* Undo / Redo

Stacks

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

Operations

01. Add (x) ⟶ st.push(x)

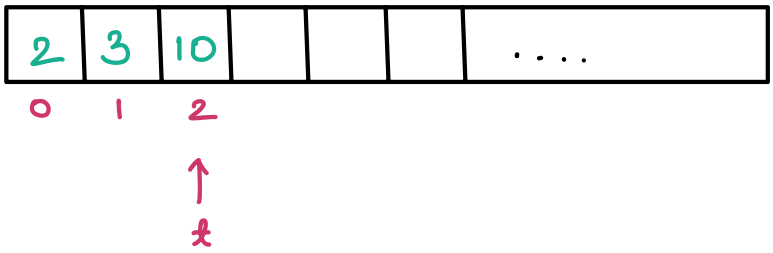
02 Remove ( ) ⟶ st.pop( )

03. Accessing topmost ele ⟶ st.peek()/st.top();

04. size ( ) ⟶ size of the stack

} O(1) time

* Implement Stack using an array



push(2)

push(3)

push(8)

pop();

peek(); ⟶ 3

push(10);

$t = -1 \not\rightarrow \not 0 \not\rightarrow \not 1 \not\rightarrow 2$

// → dynamic array

```
t = -1
void push(x){
    t = t + 1
    A[t] = x;
}
```

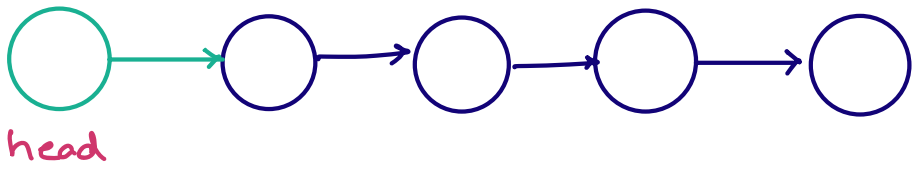
```
int pop(){
    if(t == -1){
        return -1
    }
    k = A[t]
    t = t - 1
    return k;
}
```

```
int peek(){
    if(t == -1){
        return -1
    }
    else
        return A[t];
}
```

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

∀ operation TC: O(1)

* Implement using Stack using LinkedList



Stack → Last In First Out

* Insert at tail → O(1)

  Deletion at tail → O(n)

* Insert at head → O(1)

  Deletion at head → O(1)

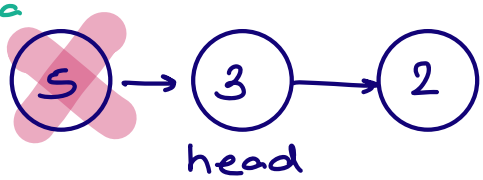
* push(2)

  push(3)

  push(5)

  peek() → return head.data

  pop() → 5



size() → maintain a counter

```
void push (x)
    Node nn = new Node(x);
    if (head == null)
        head = nn;
    else {
        nn.next = head;
        head = nn;
    }
    cnt = cnt + 1;
}
```

```
int pop()
    if (head == null) return -1
    else {
        rv = head.data;
        head = head.next;
        cnt = cnt - 1;
    }
    return rv;
}
```

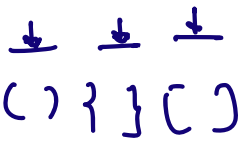
```
int size() {
    return cnt;
}
```

```
int peek() {
    if (head == null) return -1
    else return head.data;
}
```

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



str = ( )( )(( )) → Valid

str = (( )( )) → Invalid

str = (( ))( ) → Invalid

str =)( )(→ Invalid

* ( )[( )]{ }

* (}

* ( )[( )[

* While traveling from L→R

→ # open '(' ≥ # closing ')' → at all points

```
open = 0      close = 0

for (i=0; i<n; i++){
    ch = str[i]
    if (ch == '(') open++;
    else close++;
```

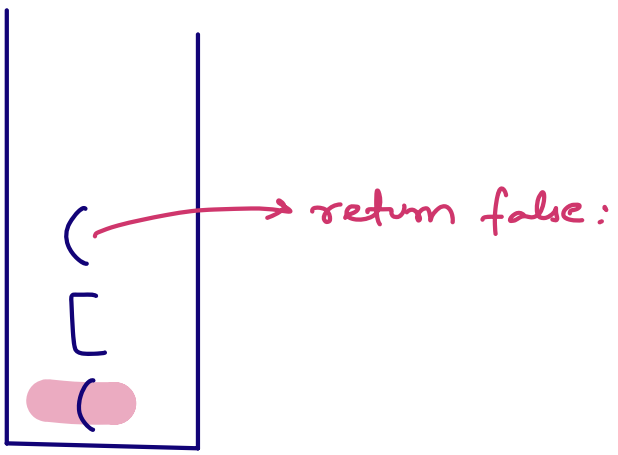
TC : O(n)

SC : O(1)

```
        if (close > open) return false;
    }
    return (open == close);
```

Str = ( ) [( ]) → return false

Idea → ∀ closing brackets → check if it is matching with last opening bracket




```
for (i=0; i<n; i++){
    char ch = str[i];
    if (ch=='(' || ch=='[' || ch=='{')
            st.push(ch);
    else if (ch==')')
    {
        if (st.size()==0 || st.peek()!='(') return false;
        else st.pop();
    }
    else if (ch==']')
    {
```

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

TC: O(n)

SC: O(n)

Str = ( ))) → Invalid

Str = { } [ ) → Invalid

Str = ( ) { }

Str = ((( )

10:10 pm → 10:20 pm

* Double Trouble

Given a string Str. Remove equal pair of adjacent characters. Return the ans without adjacent duplicates

S = abbd ⟶ ad

S = abccbde ⟶ abbde ⟶ ade

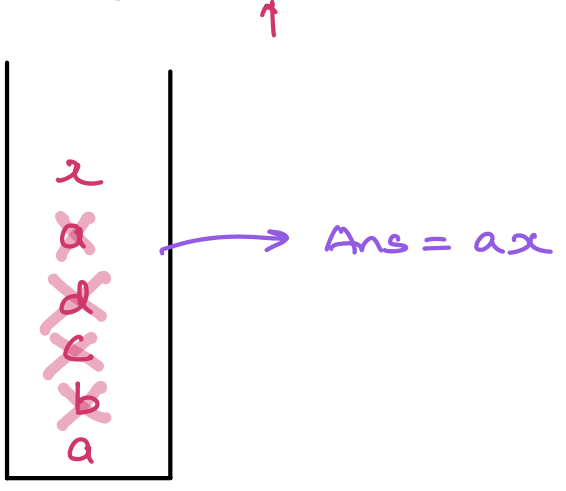
S = abbbd ⟶ abd

S = abba ⟶ " "

S = abbcbbcacx ⟶ cx

Idea → For current char, we need to check last travelled char

Str = abcddcaabx



Traverse string

if (curr = st.peek())

Yes

No

st.pop()

push(ch)

Build string

```
String ans = " ";
while (st.size()>0){
    char ch = st.pop();
    ans = ch + ans;
}
```

* Infix expression

operand1 operator operand2

1 + 2

a + b

a * b

Postfix expression

operand1 operand2 op

1 2 +

a b +

a b *

a + (b * c)

a + bc*

a bc* +

(a * b) - c

ab* - c

ab* c -

$a + (b - c)$

$a + bc-$ ⟶ $a\ bc-\ +$

* $10 + 3 * 4 - 7$

$10 + 34* - 7$

$10\ 34* + - 7$ ⟶ $10\ 34* + 7 -$

* For a given postfix expression, evaluate & return its answer $a\ b +$

01. $7\ 3 +$ ⟶ $7 + 3 = 10$

$3\ 5 + 2 - 2\ 5 * -$

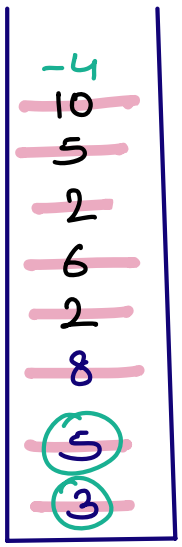
$3\ 5 + 2 - 10 -$

$8\ 2 - 10 -$

$6\ 10 -$ ⟶ -4

∀ operator → last 2 operands on which this operation has to be performed

3 5 + 2 − 2 5 * −



a b
3 + 5 = 8

8 − 2 = 6

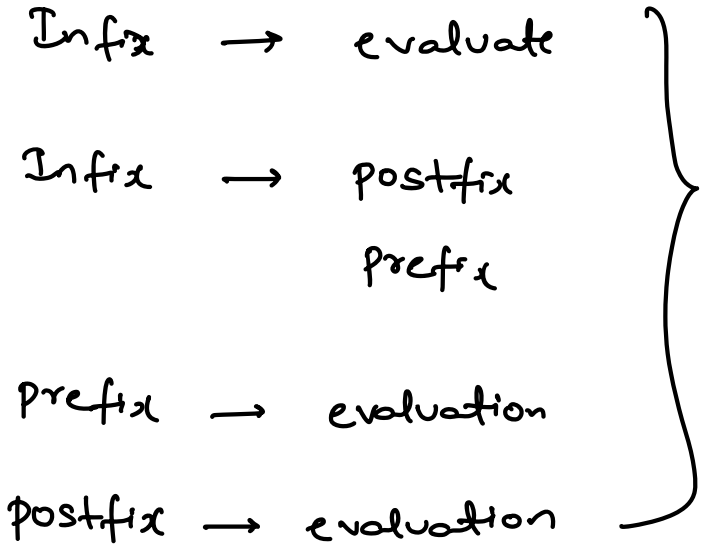
2 * 5 = 10

6 − 10 = -4

```
Stack < I > st = new stack<>();

for (i=0; i<n; i++){

    char ch = str[i];

    if (ch == operators)
    |
    |    get the 1st popped ele = b
    |
    |    get the 2nd popped ele = a
    |
    |    st.push(a (op) b)
    |
    3
    else → st.push(ch);
}
return the last operand standing in stack.
```

TC: O(n)

SC: O(n)


Infix → evaluate
Infix → postfix
prefix
prefix → evaluation
postfix → evaluation