Stacks 1

Good Evening

Todays content

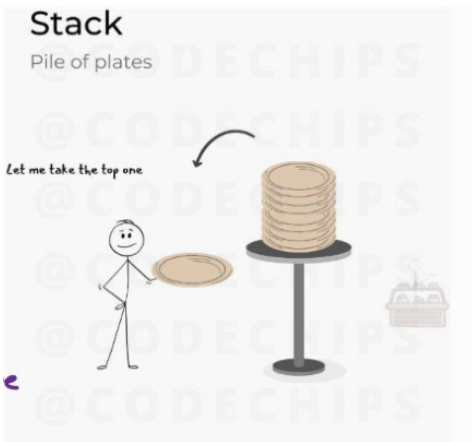
01. Stack basics
02. Implementation of Stack
    - → Using Arrays
    - → Using Linkedlist
03. Balanced brackets
04. Double trouble
05. Postfix Evaluation

Stack → Linear data structure that stores the data in a sequence: bottom to top

→ Last In First Out (LIFO)

* Real World Example

01. Book stack
02. Piles of plates
03. Idli cooker
04. Cricket stump made up of bricks



04. Recursion stack

→ All the function calls get stored one over the other in recursive stack

05. Browser History

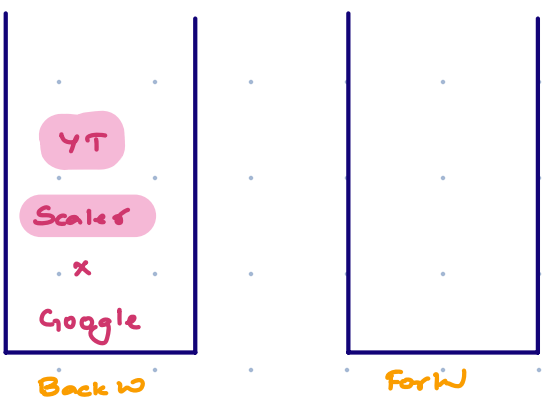
Google → x → Scaler → YT

Backw

Backw

Forw

Forw



06. Undo / Redo

Scenerio

In a **Flipkart warehouse**, boxes are stacked one over another. Each box is tagged with a weight, and for safety, heavier boxes must be placed below lighter ones. Workers need a system to check if adding a new box on top is safe.

Workers want to perform two kinds of operations:-

1. Type **ADD**: a new box of some weight on the top
2. Type **REMOVE**: the topmost box

Query Type	Value	Answer
ADD	10kg	True
ADD	5kg	True
ADD	12kg	False
REMOVE	-	
ADD	6kg	True

Remove (Boxes are empty) False

Stack Declaration in Java

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

Operations

01. Add(x) ⟶ st.push(x)

02. Remove ⟶ st.pop();

03. Access topmost element ⟶ st.peek();

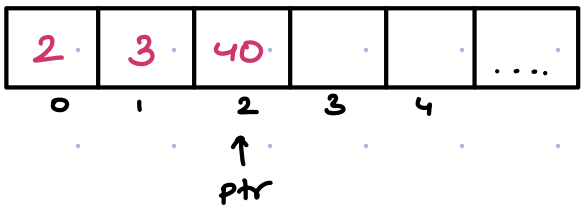
04 size ⟶ st.size();

05 isEmpty ⟶ st.isEmpty();

TC: O(1)

* Implementation of Stack

01. Using Array



push(2)

push(3)

push(20)

pop() ⇒ 20

push(5)

peek() ⇒ 5

pop() ⇒ 5

push(40)

size()

ptr = -1 → 0 → 1 → 2 → 1 → 2

A[] = new int [sz];

ptr = -1

```
void push(x){
    if (ptr+1 == sz){
        return
    }
    ptr = ptr+1
    A[ptr] = x;
}
```

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

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

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

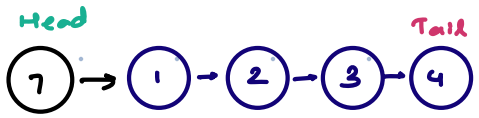
Note

Underflow → Remove elements from empty stack

Overflow → Add elements in full stack

* Implementation of Stack using Linkedlist

LIFO



pop()

pop()

push(7)

pop()

Insert at head → O(1)

Remove at head → O(1)

Insert at tail → O(1) ✓

Remoual at tail → O(n)

```
int count = 0

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

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

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

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

10:23 pm → 10:33 pm

Questions

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 = ( ) [( )] { }

Eg 2 = (( ) }

Eg 3 = (}) {

Eg 4 = ({) }

Eg 5 = ([{] })

Eg 6 = { [[ ] { }] }

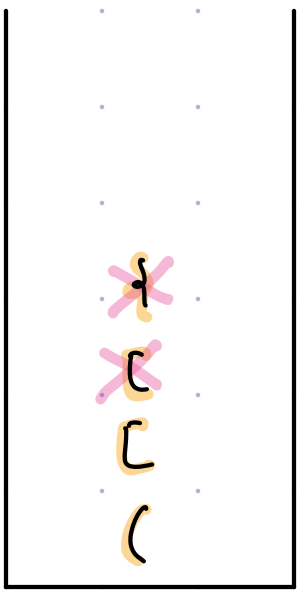
Approach → For an expression to be valid, its subexpression should also be valid

∀ for all the closing brackets, we will match them with opening brackets of same type & then remove it.

Eg 6 = ([[ ] { }) }

⇓

No round opening bracket → False



Stacks

See if we need to access the last iterated element.

```
Stack < Character > st

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

    char ch = str.charAt(i)                    str[i]

    if ( ch=='(' || ch=='{' || ch=='[') st.push(ch);

    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;
    }
    else if ( ch=='}'){

        if ( st.size()!=0 && st.peek()=='{') st.pop();

        else return false;
    }

}

return (st.size()==0)        // when extra opening brackets
                                (((
```

extra closing brackets
))

* Double Trouble

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

S = "abbd" → "ad"

S = abccbe → "abbe" → "ae"

S = abbbd → "abd"

S = abba → "aa" → ""

S = abbbdb → "abdb"

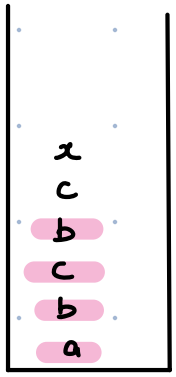
S = abbcbbcacx

= "accacx" ⇒ "aacx" → "cx"

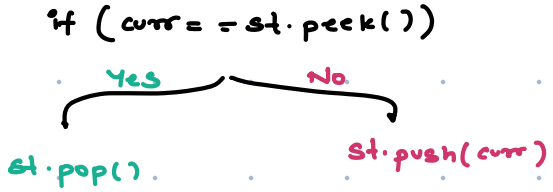
Approach

→ every character, i need to look at the previous travelled character

S = " a b b c b b c a c x "



Ans = "cx"



```
Stack < character > st ;

for ( i=0; i<n; i++){
    char curr= str. charAt(i);
    if ( st. size ()==0 || st.peek() != curr)
        st. push(curr);
    else if ( curr == st. peek()) st.pop();
}
```

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

```
StringBuilder sb = new StringBuilder ();

while (st. size ()>0){
    char ch = st. pop();
    sb. append (ch);
}

return sb. reverse (). toString();
```

* Evaluation of Expression

Infix expression

operand1	operator	operand2
1	+	2

Postfix expression

Operand1	Operand2	Operator
1	2	+

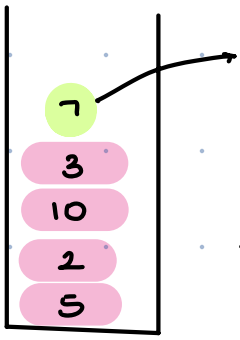
01. a + b — ab +

02. a * b — ab *

03. a + b * c
    ↓
    a + bc *

    a bc * +

04. 5 2 * 3 −

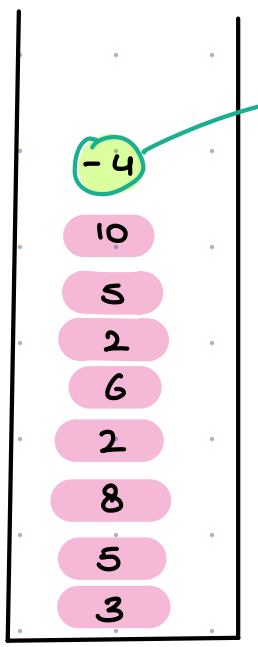


Ans = 7

b = 2
a = 5
val = 5 * 2

b = 3
a = 10
val = 10 − 3 = 7

05 3 5 + 2 − 2 5 * −



Ans = −4

b = 5
a = 3
val = 3 + 5

b = 2
a = 8
val = 8 − 2

b = 5
a = 2
val = 2 * 5

b = 10
a = 6
val = 6 − 10

BODMAS

```
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();
```

Hints

x = ~~3~~ 4

y = 5

z = ~~10~~ 11

++z + y - y + z + x++;

11 + 5 - 5 + 10 + (3++)

16 - 5 + 10 + (3++)

11 + 10 + (3++)

21 + (3++)

24