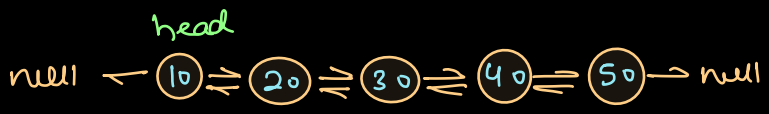
Stacks 1

~~~~~~~
Agenda
~~~~~~~

1. Revision Quiz
2. Introduction to Stack
3. Scenerio based Problem
4. Stack Implementation
5. Balanced Paranthesis
6. Remove Equal Pairs
7. Postfix Evaluation

Hello Everyone
Very Special Good Evening
to all of you 😊😊😊
We will start discussion
from 09:06 PM

~~~~~~~~~~~~~~
Revision Quiz
~~~~~~~~~~~~~~



Quiz 1 :
What is the primary advantage of using a doubly linked list over a singly linked list?
[X] Simplified insertion at the end → it depends if we have tail or not.
[X] Reduced memory usage → it increase
[✓] Easier traversal in both directions
[X] Faster access to elements

Quiz 2 :
When deleting a node from a doubly linked list, which pointers need to be updated?
[X] Only the next pointer of the previous node
[X] Only the prev pointer of the next node
[✓] Both the next pointer of the previous node and the prev pointer of the next node
[X] No pointers need to be updated

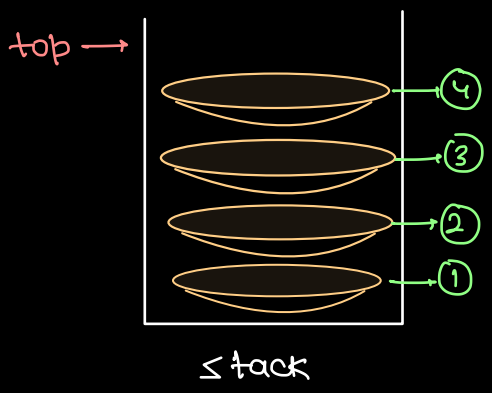
Quiz 3 :
What data structures can be combined to efficiently implement an LRU cache?
[ ] Array and Stack
[✓] Doubly Linked List and HashMap
[ ] Stack and Queue
[ ] ArrayList and TreeMap

→ Hashmap
→ Doubly LL

Introduction to Stack :

* A stack is a linear data structure that stores information in a sequence, from bottom to top.

* The data items can only be accessed from the top and new elements can only be added to the top, i.e it follows LIFO (Last In First Out) principle.



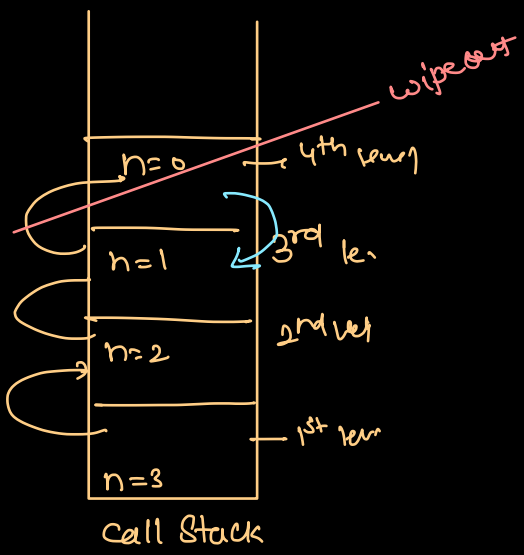
* Remove top plate
* Insertion possible at top position.

Quiz :
What is the most common application of a stack?
[✓] Evaluating arithmetic expressions
[✓] Implementing undo/redo functionality
[✓] Used in recursion internally
[✓] All of the above → correct

Dry Run of Recursion:

```
void fun( ——— ) {
    if(n==0){
        return;
    }
    fun( ——— );
    task
    fun( ——— );
}
```

3



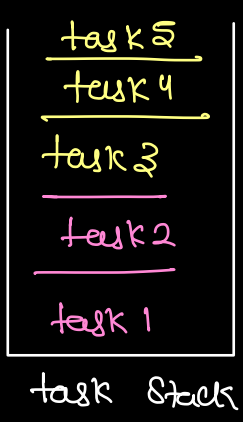
→ insertion happens on top
→ Removal from top

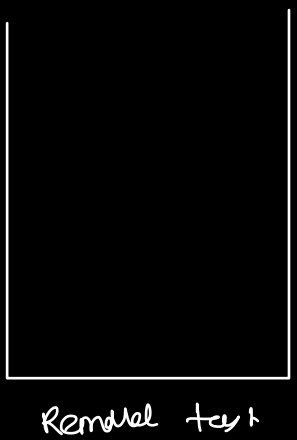
LIFO → Stack

Redo / Undo function

Browser → site movemet

task 1 → task 2 → task 3 → task 4 → task 5





Postfix expression:

7 + 5 * 3 → 36 → A ✗

→ 29 → B ✓ → Becse of priority of operator

infix Expression

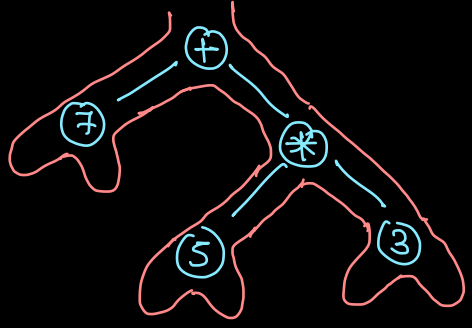
machine will convert infix expresion to postfix expresion.
then it perform calculation.

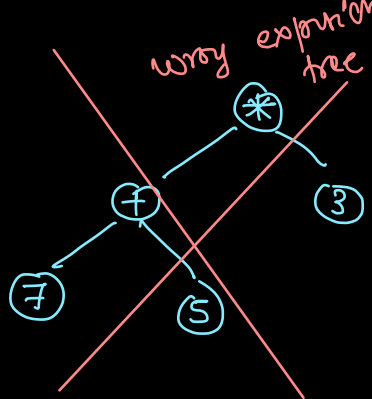
priority of operator already handle.

Expression tree:

(in)

Expression: 7+5*3





infix → 7 + 5 * 3

postfix → 7 5 3 * +

↳ operators are already arranged according top.

Flipkart Warehouses Box management

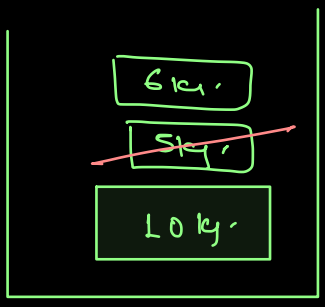
Scenerio

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:-

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

Example

Query Type	Value	Answer (true/false)
ADD	10kg	true
ADD	5kg	true
ADD	12kg	false
REMOVE		5kg. Box is remove
ADD	6kg	true



Add → * if stack is empty or top weight is greater than curr
→ add that box
→ Return true
* else → Return false

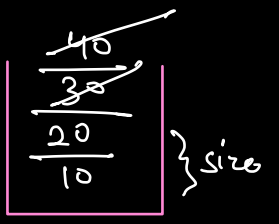
Remove → * if stack is not empty
Remove top most

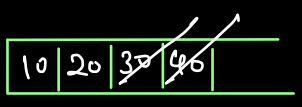
Operations on Stack

* Insertion → push(data x)
* Removal → pop()
* Accessible element in Stack → peek() (top element)
* if something is available or it is Empty → isEmpty()
* size → size()

T.C. of Stack

push	O(1)
pop	O(1)
top	O(1)
isempty	O(1)
size	O(1)





Implementation of Stack with Arrays

* void push(x)
* datatype pop()
* datatype top()
* Int size()
* boolean isEmpty()

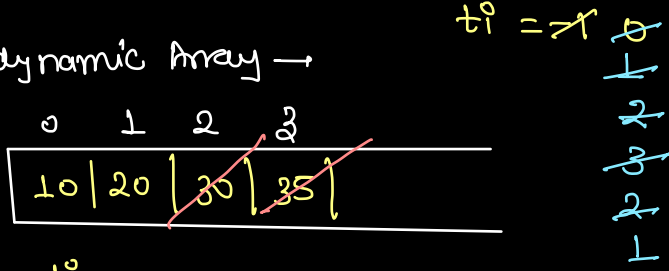
operations:

push(10); ✓
push(20); ✓
push(30); ✓
push(35); ✓

top() ⟶ 35
pop() ⟶ 35
pop() ⟶ 20
top() ⟶ 20
isEmpty() ⟶

Stack

dynamic Array →



0 1 2 ~~3~~ ~~2~~ 1

```
int ti = -1;
void push(int x) {
    ti++;
    list.add(x);
}

int top() {
    return list.get(ti);
}
```

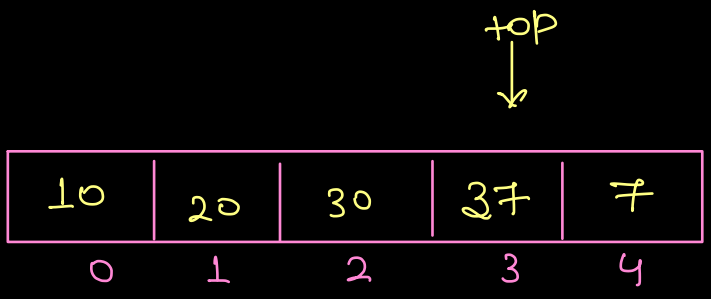
⇒ list[ti]

```
int pop() {
    // remove elmnt
    int rem = list.remove(ti);
    // Decrease top index
    ti--;
    return rem;
}

boolean isEmpty() {
    return ti == -1;
}
```

Underflow and Overflow situation:

Using fix size array:

capacity of stack = 5 (Assumption):



top indx = ~~-1~~ ~~0~~ ~~1~~ ~~2~~ ~~3~~ 3 ~~3~~ ~~4~~

```
push(10); ✓
push(20); ✓
push(30); ✓
push(35); ✓

top() ⟶ 35
pop() ⟶ 35
push(37); ✓
push(7); ✓
push(9); ✓ ⟶ overflow situation
top() ✓ ⟶ (7)
push(11); ⟶ overflow
pop() ⟶ (7)
top() ⟶ (37)
isEmpty() ⟶ false
```

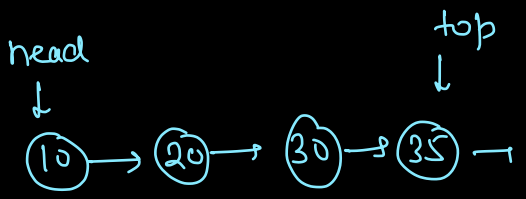
```
int top = -1;
void push(int x) {
    if(topindx == arr.length-1) { overflow situation }
    topindx++;
    arr[topindx] = x;
}

int top() {
    if( isEmpty() == true) { return -1;}
    return arr[topindx];
}

int pop() {
    if( isEmpty() == true) { return -1;}
    int rem = arr[topindx];
    topindx--;
    return rem;
}

boolean isEmpty() {
    return topindx == -1;
}
```

Implementation of Stack using Linked List

* void push(x)
* datatype pop()
* datatype top()
* int size()
* boolean isEmpty()

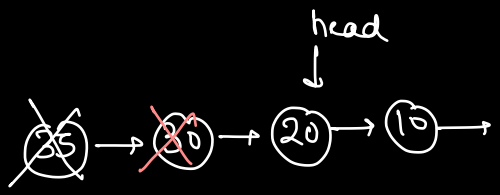
operation:

push(10); ✓ addLast(10)
push(20); ✓ addLast(20)
push(30); ✓ addLast(30)
push(35); ✓ addLast(35)



pop() → O(n) T.C.

top() → 35 → getLast()
pop() → (tail-1) is required → Iterate till second last
pop() → O(n) → removeLast() → O(n)
top()
isEmpty()



push(10); ✓ addFirst()
push(20); ✓ addFirst()
push(30); ✓ "
push(35); "

top() → getFirst() → 35
pop() → removeFirst → O(1) → 35
pop() → " → O(1) → 30
top() → 20
isEmpty() → false.

head
null

```
int sz = 0;
Node head;

void push(int x) {
    Node nn = new Node(x);
    nn.next = head;
    head = nn;
    sz++;
}

int pop() {
    if(head == null) { return -1; }
    int rem = head.data;
    Node temp = head;
    head = head.next; ⟶ for Java
    free(temp); ⟶ for others.
    sz--;
    return rem;
}

int top() {
    if(head == null) { return -1; }
    return head.data;
}

boolean isEmpty() {
    return head == null;
}

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

10:54 – 11:00 pm
Break.

Balanced Paranthesis

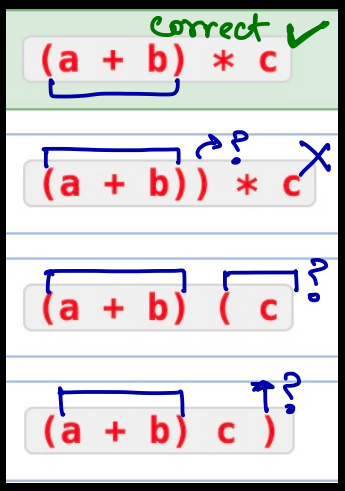
Check whether the given sequence of parenthesis is valid ?

eg1: { [( ) (( )) [[{ ( ) }]]] } → Balanced bracked OR valid

eg2: { [( ) (( ) } [ ] ( ) → unbalanced OR invalid pair.

technical aspect of this problem:

Infix: → 5 + [(a+b) * (3+2] → if invalid bracket hard to solve.



Pairing of bracket:

{ → } , [→] , (→)

eg: [{()()(())}{}]

count of opening

[→ 1
{ → 2
(→ 4

count of closing

] → 1
} → 2
) → 4

if open's count < similar closing count is matching problem solved ?

eg: [[()]](()){{}} → false.

unmatched

op

[→ 1
{ → 1
(→ 3

cl.

] → 1
} → 1
) → 3

eg: [{()()(())}{}] → Seq. of bracket

opening → [, {, (
closing →], },)

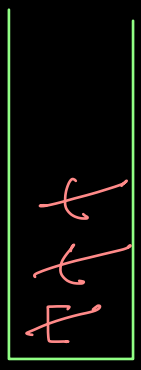
opening → push in the stack

closing → valid counter on top of stack (pop)
→ invalid counter → Return false
→ stack is empty. → Return false.

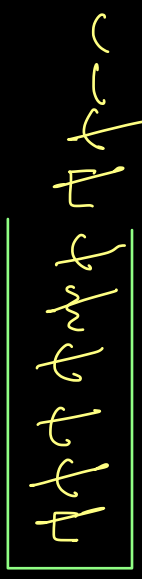
→ in end if nothing is in stack return true.

eg: [()](){}

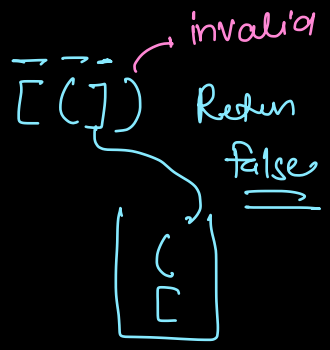
Return false.



eg: [()(())]{()}[()](((



Return false




```
boolean isBalanced(String str){
    Stack<Character> st = new Stack<>();
    for(int i=0; i<str.size(); i++){
        char ch = str[i];
        if( ch=='[' OR ch=='(' OR ch=='{'){
            st.push(ch);
        }
        else if( ch==']' OR ch==')' OR ch=='}' ){
            if(st.size()==0) return false;
            if( st.peek() is not same as corresponding opening bracket)
                return false;
            st.pop();
        }
    }
    return st.size()==0;
}
```

T.C: O(n)

S.C: O(n)

Remove equal pair of consecutive elements till possible

Given a string, remove equal pair of consecutive elements till possible

eg → a b c d d c

Remove them

⇓

a b c c

Remove them

⇓

a b ⟶ ans

eg: a b c c b d d d b c a a b

a b b d d d b c a a b

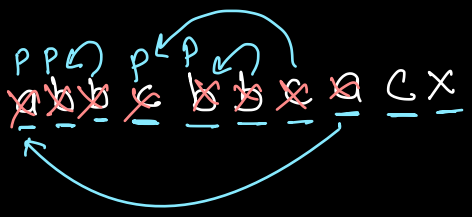
a d d d b c a a b

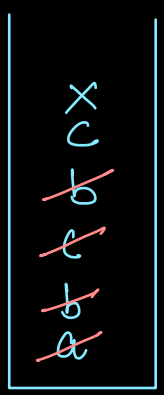
a d b c a a b

a d b c b ans.

Quiz

eg:





x c Reverse → c x

```
String doubleCharacterTrouble(String str){
    Stack<char> st ——————
    for(int i=0; i<n; i++){
        char ch= str[i];
        if(st.size()==0 OR st.peek()!=ch){
            st.push(ch);
        }
        else{
            st.pop();
        }
    }
    // Remove char from stack & make a str
    // Reverse that string
    return that string.
}
```

Java → StringBuilder

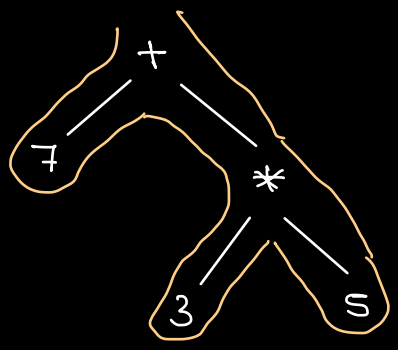
T.C: O(n)
S.C: O(n)

Evaluate Postfix Expression

Given a postfix expression, evaluate it.

infix → 7 + 3 * 5

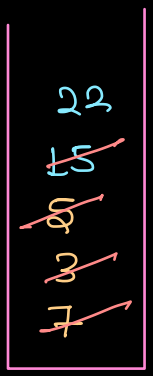
infix Expression → tree



postfix → 7_3_5_*_+_

postfix : 7 3 5 * +

operators are already arranged according to the priority.



int - Stack

operand → push it in stack

operator →

*

val2 = 5
val1 = 3

evaluate (3, 5, *) ⇒ 15

st.peek() → or ⇒ 22

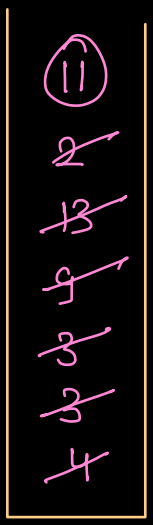
+

val2 → 15
val1 → 7

evaluate (7, 15, +) ⇒ 22

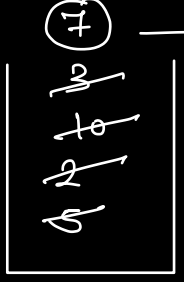
[4, 3, 3, *, +, 2, -]

op =	*	+	-
val1 = 3	4	13	
val2 = 3	9	2	



st.peek() → ans.

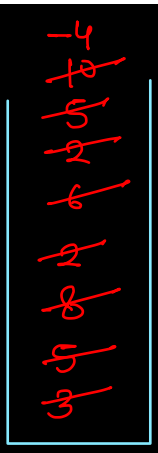
5 2 * 3 -



val2 = 2	3
val1 = 5	10

Evaluate the given postfix expression:

3 5 + 2 - 2 5 * -



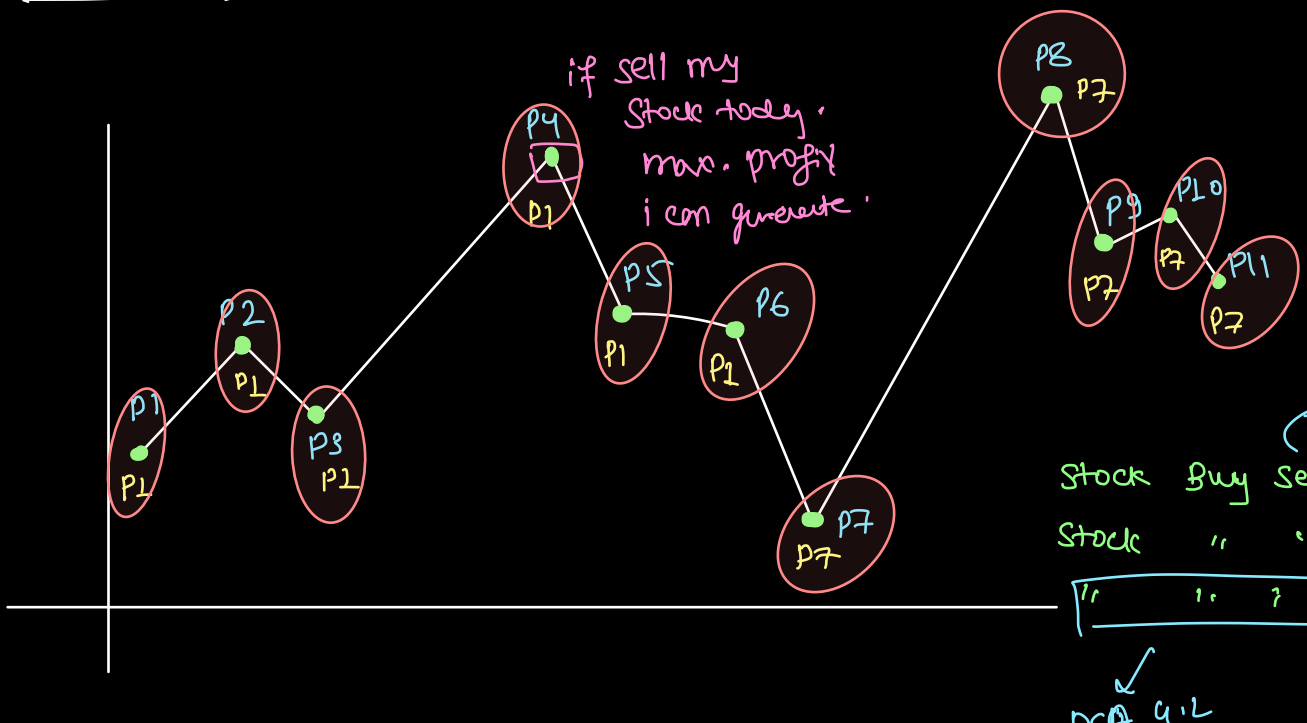
op = +	-	*	-
val1 = 3	8	2	6
val2 = 5	2	5	10

↳ st.peek() → -4

operand → push

operation → val2 = st.pop()
val1 = st.pop()
rs = evaluate(val1, val2, op)
st.push(rs)

Doubt session:



using.

Stock Buy Sell → (1)

Stock " " → (2)

" " " → (3)

DP

DSA Q:2

Problems:- pepcoding. → Array & String. → Pick the Problem. Solve it on any platform. (leetcode gfg.) → Use videos

Searching & Sorting.

Stack

TRY To Solve:

I need help in binary search for three numbers sum and difference one how to solve that. example find the 5th sum value from an array [3,4,5,1,2,6,7,9,8,10] answer will be 14.