Stacks

Defn :-

- 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.

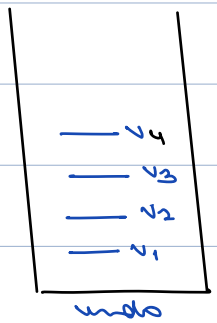
1) Pile of Plates.

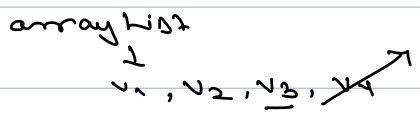
2) Stack of chairs.

Algorithmic Examples :-

1) Recursion,

2) Undo - Redo.





Arraylist
↳ add
↳ remove.

Operations on Stack

1) push (data) ⟶ inserts a new element into top of the stack.

2) pop() ⟶ Removes an element out of the stack.

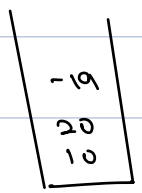
3) peek() → gives access to top element of stack, without removing it.

↓
top()

```
- st -
int x = st.peek();
```

4) isEmpty() → checks stack is empty or not.

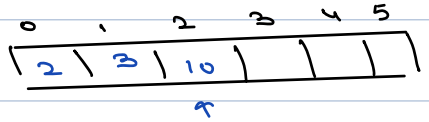
```
Stack st = new Stack();
st.push(10);
st.push(20);
st.push(-19);
int x = st.peek();
print(x) → -19
```

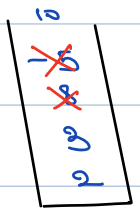


→ Implementation of Stacks using Arrays.

arr -> 6.

top = -1 0 1 2





push(2)

push(3)

push(8)

Pop(); ⟶ 8

Push(-5)

Peek() ⟶ -5

Pop() ⟶

Push(10)

```
class Stack {
    private int arr[];
    private int top;

    Stack() {
        arr = new int[10];
        top = -1;
    }

    void push(int x) {
        top++;
        arr[top] = x;
    }

    int pop() {
        int x = arr[top];
        top--;
        return x;
    }
}
```

int peek () {

return arr[top];

}

boolean isEmpty () {

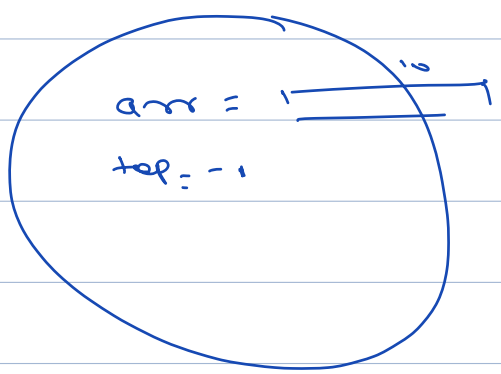
if (top == -1) { return True }

return false;

}

}

Stack st = new stack ();



Overflow

```
void push(x){
    // Whenever our counter reaches to the size of the array
    // It means stack is already full
    if(t >= A.size())
        return;
    t++;
    A[t] = x;
}
```

Underflow :-

- Underflow means when we try to perform pop operation or try to access the element of stack but there are none. Again we have to introduce conditions during pop and top operation.

```
void pop(){
    if(isEmpty()) return;
    t--;
}
```

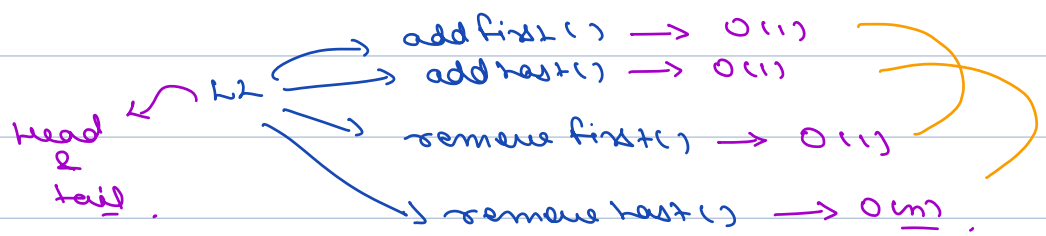
peek

```
int top(){
    if(isEmpty()) return -1;
    return A[t];
}
```

Problem :-

We have to predefine the size of stack to create array. To overcome this problem we can create a dynamic array which can grow or shrink at runtime according to need.

Implementing Stacks using LL :-



addFirst & removeFirst

push(2) ✓

push(3) ✓

push(8) ✓

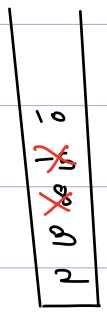
pop(); → ✓

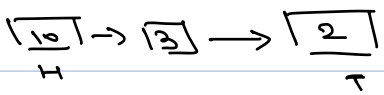
push(-5) ✓

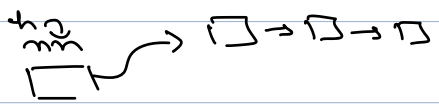
peek() → ✓

pop() → ✓

push(10) ✓







```
void push(data){
    new_node = Create a new Node with 'data'
    new_node.next = head
    head = new_node
    // Increment size
    t++
}
```

→ internal variable for size.

```
void pop(){
    if( isEmpty()) return;
    head = head.next
    // Decrement size
    t--
}
```

→ Peek()

```
int top(){
    if( isEmpty()) return -1;
    return head.data;
}
```

Ques Check given, sequence of parenthesis is valid. { , [, (

e.g -> ( ) [{ } ( )] → True.

( ) [{ (})] → false.

({ } [ ] → false.

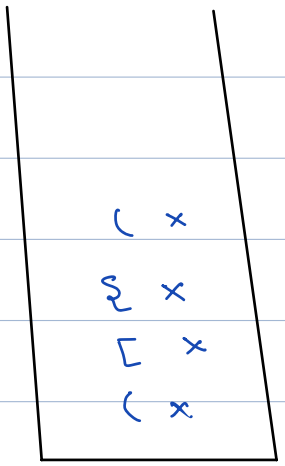
) ([ ] → false

({ [ ] }) → True

→ closing Bracket → opening brkt should be of same type.

→ All opening Brts, should be closed.

( ) [{ } ( )]



→ True.

(x
[
{
(
→ false.

→ false.

(x
{ x
[
→ false.

Traverse complete string

open bracket → push(x)

close →

check top of stack ();

match () → pop();

no match () → return false;

}

stack Empty → return True;

else return false

T.C → O(n)

S.C → O(n).

Ques :-

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

e.g → abbc → ac

abccbde → abbde → ade

abbbcccca → abca.

e.g,

a bb c bb c a cx

↓

a c c a cx

↓

a a cx

↓

cx .

* characters,

if top matches, → pop();
else push(x)
↳ reverse what you got out of stack.

T.C → O(n), S.C → O(n).

Post fix Notation

Infix notation	Post fix notation
2 + 3	2 3 +
operand operator operand.	↓ operand1 operand2 operator.
a + b	ab+
(a * b) - c	ab * c-

e.g.,

a + b − c * (d + e)

↓

a + b − c * de+

↓

a + b − cde+*

↓

ab+ − cde+*

↓

ab+ cde+* −

e.g, 2 * (3 − (6 / 2)) → 2 * (3 − 3)

↓

2 * (3 − 6 2 /)

↓

2 * 3 6 2 / −

↓

2 3 6 2 / − *

2 * 0

↓

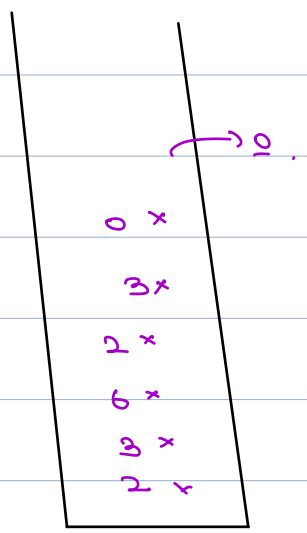
0.

Benefits of postfix

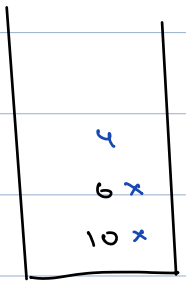
1) doesn't have any bracket
2) Easier to Evaluate → O(n).

Evaluate Postfix Expression

2 3 6 2 / − *

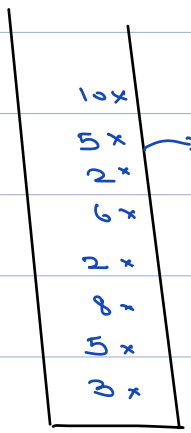


eg) 10 − 6 → 10 6 =



10 − 6 => 4

e.g 2 3 5 + 2 − 25 * −



→ −4 Ans.

5 x → 10 6 − 10 = −4

8 − 2 = 6

→ r.c=0, c.c=0.

Travel the Expression

```
if (operand) {
    push(expr);
}
else {
    x = pop();
    y = pop();
    push(y expr x);
}
```

whatever is left in stack is ans.