Stack → data structure which works on the principle of LIFO (Last In First Out)

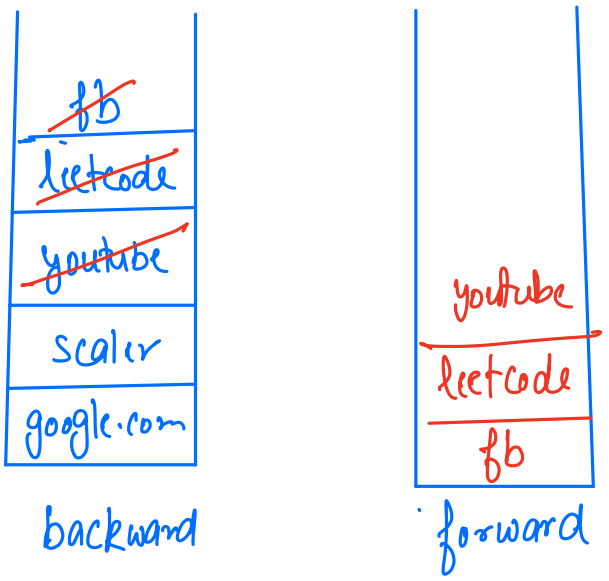
Eg - stack of books

Eg - stack of plates / CD's / chairs.

Eg - stack of functions.

Eg - Browser history

Forward and Backward Tabs



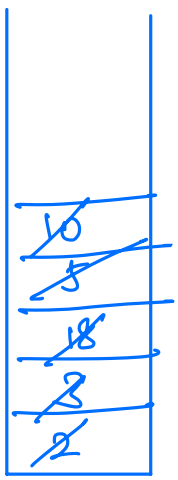
Operations in Stack

1. push(x) → insert x at the top of the stack.
2. pop() → remove the top-most element of the stack.
3. top() / peek() → get the top-most element of the stack.
4. isEmpty() → check if stack is empty or not.

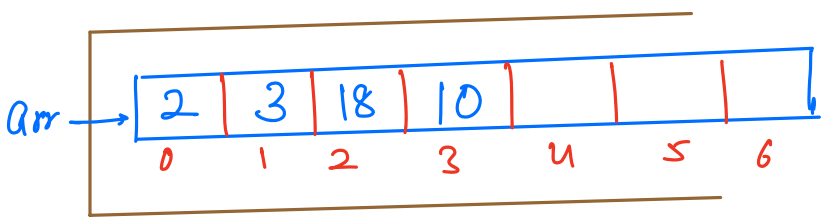
→ O(1)

Implement stack using Arrays -

push(2) ✓
push(3) ✓
push(18) ✓
push(5) ✓
pop() ✓
top() / peek() ✓
push(10) ✓
isEmpty() ✓
pop() ✓
pop() ✓
isEmpty() ✓
pop() ✓
pop() ✓
isEmpty()



tos → index of the topmost element.



tos

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

→ overflow → Don't allow the insertion

↳ Vector / ArrayList

```
int top() {
    if(tos == -1) { "stack is empty" }, return -1
    return arr[tos];
}
```

```
boolean isEmpty() {
    if(tos == -1) { return true }
    else { return false }
}
```

T.C → O(1)

```
void pop() {
    if(tos == -1) { "stack is empty" }, return
    tos--;
}
```

Implement Stack using Linked-list

→ push(), pop(), top()/peek(), isEmpty()

① push at head and pop from tail ✗

② push at tail and pop from head ✗ [LIFO discipline is not maintained]

③ ✓ push at head and pop from head

④ push at tail and pop from tail ✗ — O(N)

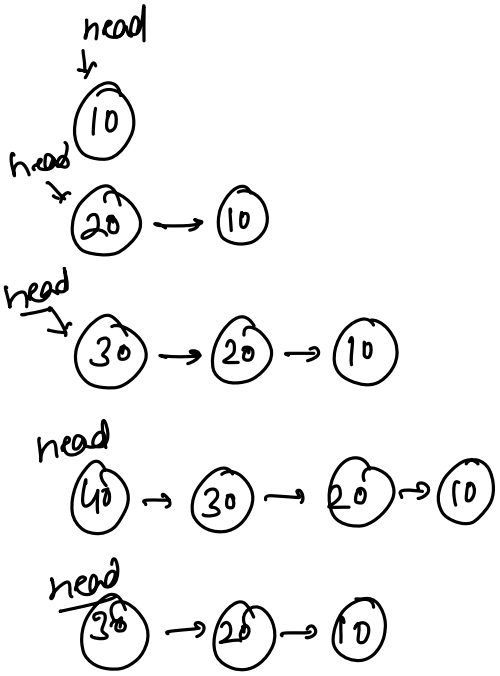
push(10) → addAtHead(10)

push(20) → addAtHead(20)

push(30) → addAtHead(30)

push(40) → addAtHead(40)

pop() → removeAtHead()



Q) Check whether the given sequence of parentheses is valid or not?

[ ], { }, ( )

Eg → ( ) [{ } ( )] → true.

( ) [ ] → true

({ }) → true.

({) } → false

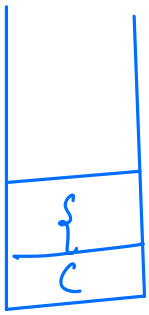
([}]) → false.

([{ } [ ]] ( )) → true.

Eg → ({ [{ } ( )] } ( )) → true.

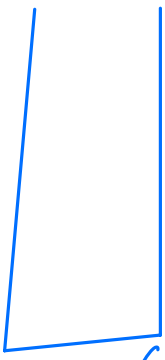
stack.

Eg - ({) } → false.



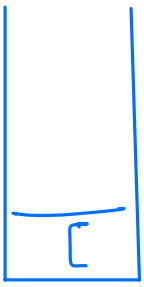
mis-match

Eg → ({ })] → false.



(more closing parenthesis)

Eg → [( ) → false.



(more - opening parenthesis)

code →

```
Stack<character> st;
for(i=0; i<N; i++){
    char ch = str[i];
    if(ch=='(' || ch=='{' || ch=='['){
        st.push(ch);
    }
    else{
        if(st.isEmpty()==true){ //more closing parenthesis
            return false;
        }
        else if( st.top() is not the counterpart of ch){ //mismatch
            return false;
        }
        else{
            st.pop();
        }
    }
}
if(st.isEmpty()==true){
    return true;
}
else{ //more opening parenthesis
    return false;
}
```

T.C → O(N)
S.C → O(N)

Double Character Trouble

Given a string str. Remove equal adjacent characters.

Return the string without adjacent duplicates.

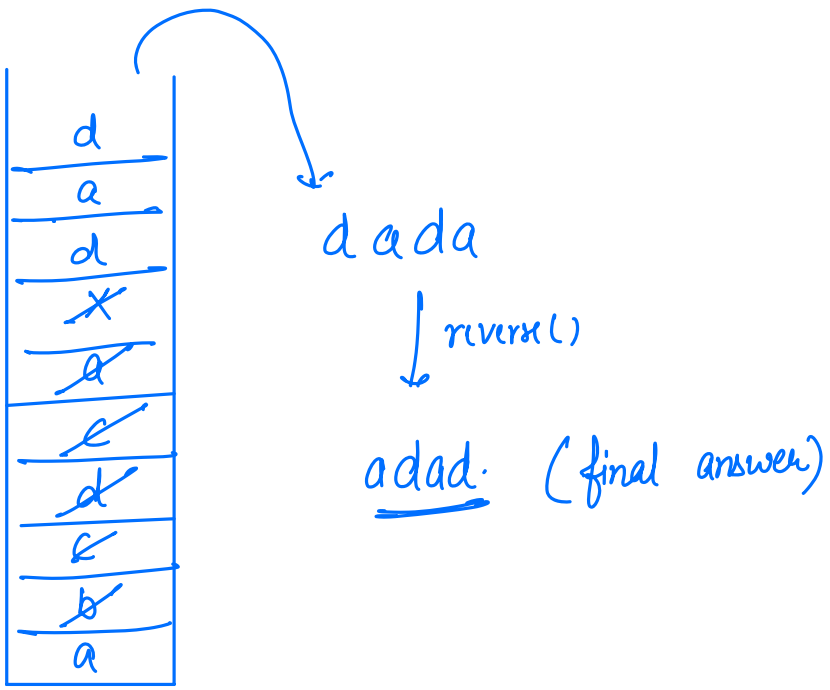
① abbd → ad

② abccbde → abbde → ade

③ abbbd → abd

④ abbcbbcacx → accacx → cx

⑤ abcdccdcaabxxdad → adad.



#code→

```
Stack<character> st;

st.push(str[0]);

for(i=1; i<N; i++){
    char ch = str[i];
    if(st.isEmpty()==false && st.top()==ch){
        st.pop();
    }
    else{
        st.push(ch);
    }
}

if(st.isEmpty()==true){ return ""; }

String ans = "";

while(st.isEmpty()==false){
    ans += st.top();
    st.pop();
}
```

→ Reverse the ans string and return it;

T.C → O(N)
S.C → O(N)

---

infix.

2+3

2+(5*3)

op1 operator op2

postfix.

2 3 +

2 5 3 * +

op1 op2 operator

Q: Evaluate given valid postfix expression.

① 10 6 - ans = 4.

② 2, 5, *, 3, - ans = 7.

10

③ 2, 5, 3, *, - ans = -13

2 15 -

④ 3, 5, +, 2, -, 2, 5, *, - ans = -4

8

6 2 5 * -

6 10 -

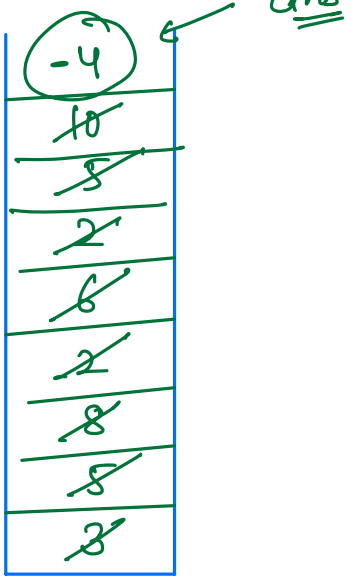
idea:- If operator, we need to have two latest operands.

⇓

Stack.

✓ ✓ ↓ ✓ ↓ ✓ ✓ ↓ ↓

arr → [3, 5, +, 2, -, 2, 5, *, -]



op2 → st.pop → 10

op1 → st.pop → 6

op1 operator op2

#code.-

```
int solution ( String [] arr , int N) {
    Stack < Integer> st;
    for( i = 0; i < N; i++) {
        str = arr[i];
        if( str is an operator) {
            int op2 = st.pop();
            int op1 = st.pop();
            res = evaluate( op1, op2, operator);
            st.push (res);
        }
        else{
            st.push( Integer( str ));
        }
    }
    return st.top();
}
```

T.C → O(N)
S.C → O(N)

```
int evaluate( int op1, int op2, String operator){
    if( operator.equals("+")){
        return op1 + op2;
    }
    else if ( operator.equals("-")){
        return op1 - op2;
    }
    else if ( operator.equals("*")){
        return op1 * op2;
    }
    else{
        return (op1/op2);
    }
}
```

- ① Searching. → Revise B.S problems — painter partition
- ② Hashing → Assigment + additional problems
- ③ Strings. → observation based problems.

Syllabus → Searching, 2P, Hashing, Strings.