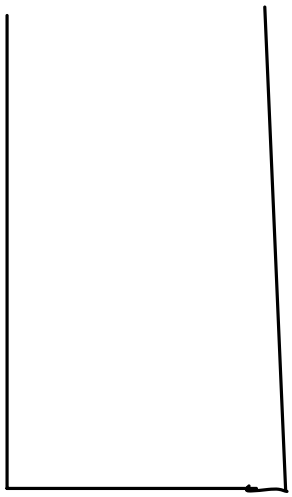
Stacks

LIFO

[Last in First Out]

1) Stack of plates
2) Stack of book
3) Stack of chairs

#



1 2 3 4

insertion order

if elements are taken out.

4 3 2 1

ORDER is reversed !

Operations supported by Stacks

1) push (x) ⇒ inserts x into Stack

2) pop () ⇒ remove the last inserted item from Stack.

3) top() / peek() ⇒ return the last inserted item.

4) isEmpty() ⇒ true if Stack is empty false if not.

5) size() ⇒ size of stack.

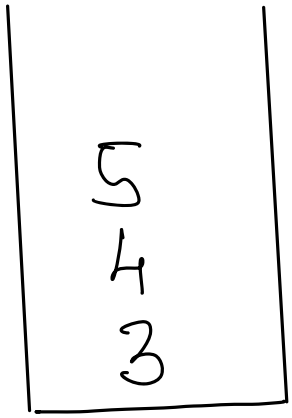
TC: O(1) for every operation

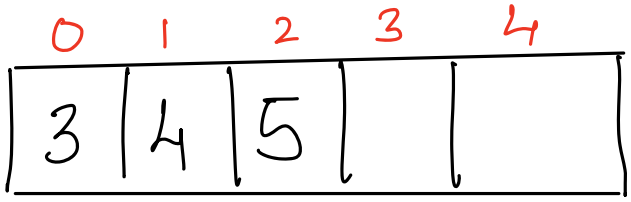
Application of Stack

- Undo & redo
- recursion
- browser back & forth
- bracket evaluation
- Equate Equation

Stack Implementation using Arrays

- push (3)
- push (4)
- top ()
- push (5)
- pop ()
- push (10)





top = 2

capacity = 5

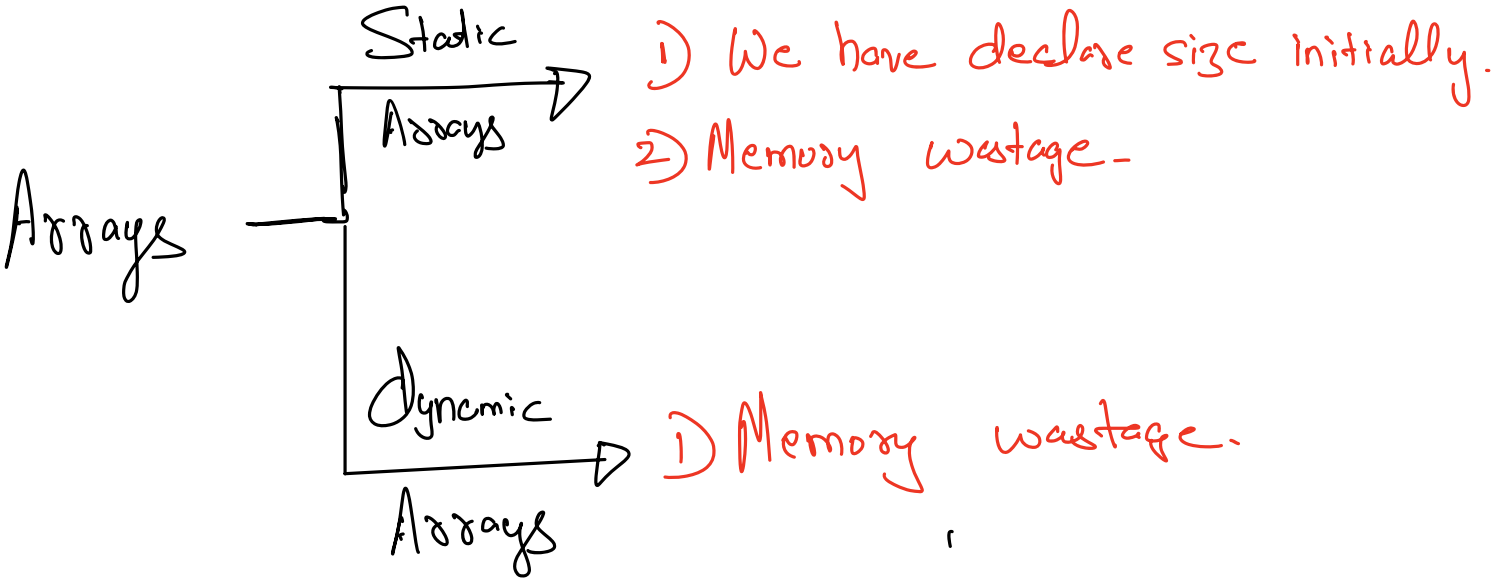
top = -1

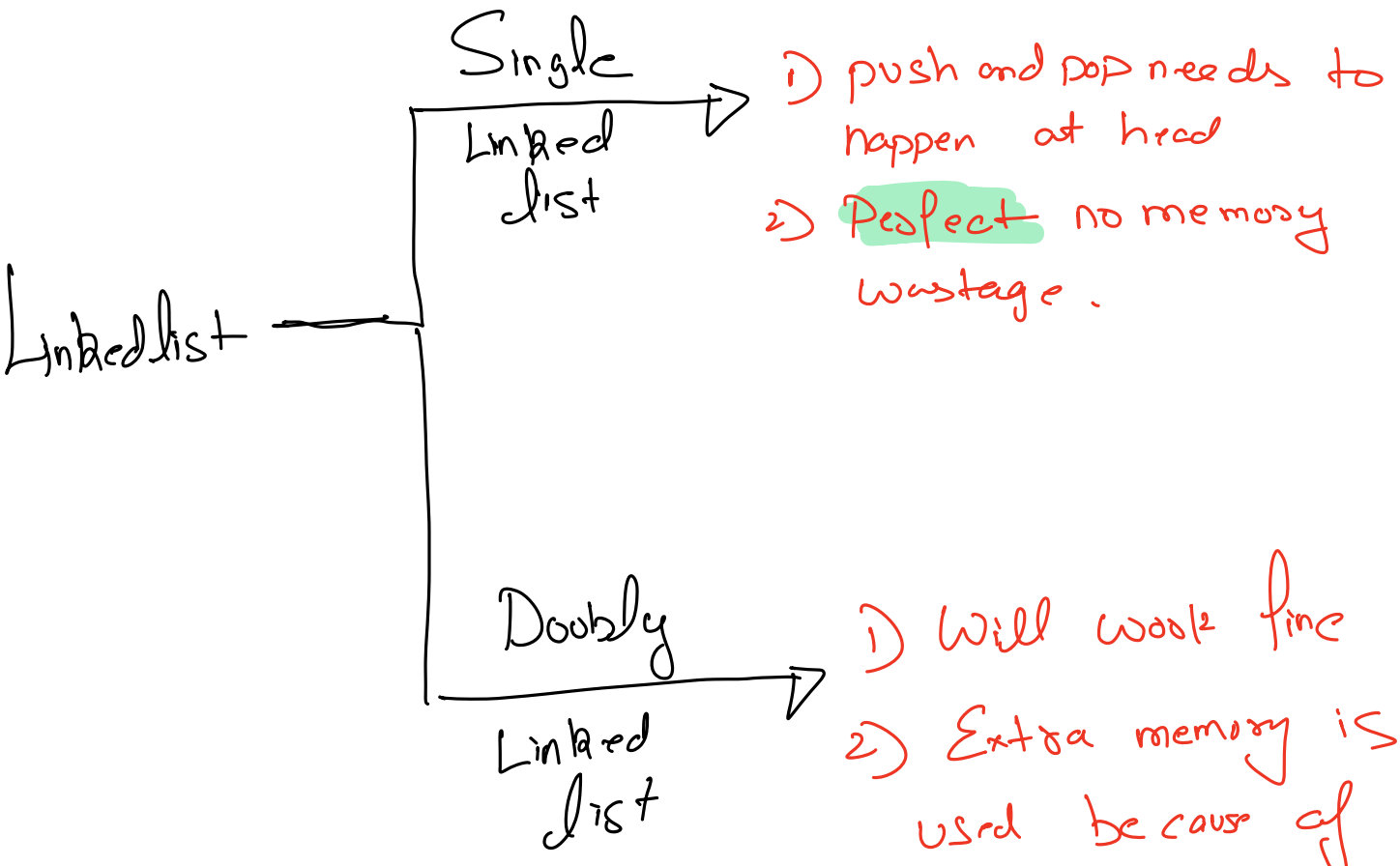
↳ array is filled till top.

```
void push (int x) {
    if (top == (capacity-1))
        return;
    top++;
    arr[top] = x;
}

bool isEmpty() {
    if (top == -1)
        return true;
    return false;
}
```

```
int top() {
    if (isEmpty())
        throw error;
    return arr[top];
}

void pop() {
    if (isEmpty())
        throw error;
    top--;
}
```

Implement using Linked linked ⇒ TODO





Q1 Check if sequence of parenthesis is valid.

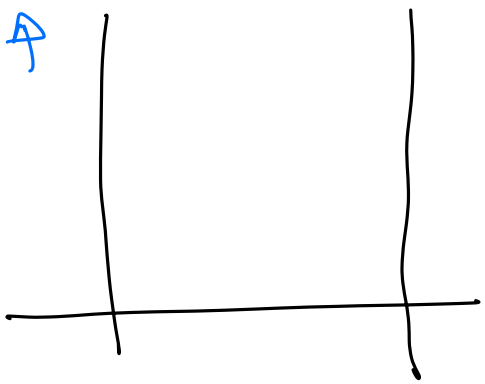
Eg1 { [ ] } ⇒ VALID

Eg2 { [ ]) ⇒ NOT VALID

Eg3 { (}) ⇒ NOT VALID

Eg4 [ ] ( ) { } ⇒ VALID

( ) [{ } ( )]



→ if you encounter an opening bracket push to stack.

→ if you encounter a closing bracket check top of stack. It should have a opening bracket of similar type.

→ If the stack is empty then sequence is valid.

Pseudo Code !

```
bool isValid (string s) {
    int len = s.size();
    Stack<char> st;
    for(int i=0; i<len; i++) {
        if (s[i]=='(' || s[i]=='{' ||
                s[i]=='[') {
            st.push(s[i]);
        } else {
            char c = st.top();
            if ( s[i]==')' && c!='(')
                return false.
            if ( s[i]==']' && c!='[')
                return false.
            if ( s[i]=='}' && c!='{')
                return false.
            st.pop();
        }
    }
    if (st.size==0)
        return true
    else return false;
```

TC: O(n)

SC: O(n)

Q2 Given a string. Remove every consecutive duplicate pair of characters until there are no consecutive duplicate pairs.

Ex1 S: a c b b c K

a c c K

a K

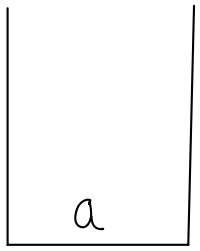
Ex2 S: a a a K

a K

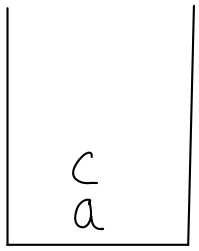
Ex3 S: a b c K K c b a d m m c

→ d c

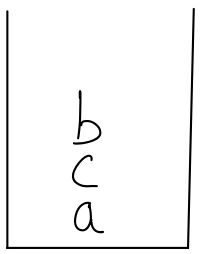
a c b b c K



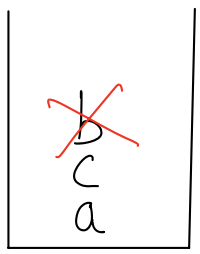
a c b b c K



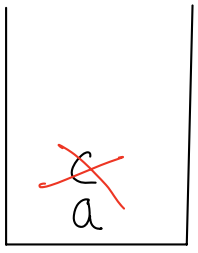
a c b b c K



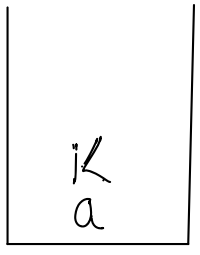
a c b b c K

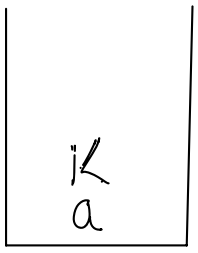


a c b b c K



a c b b c K





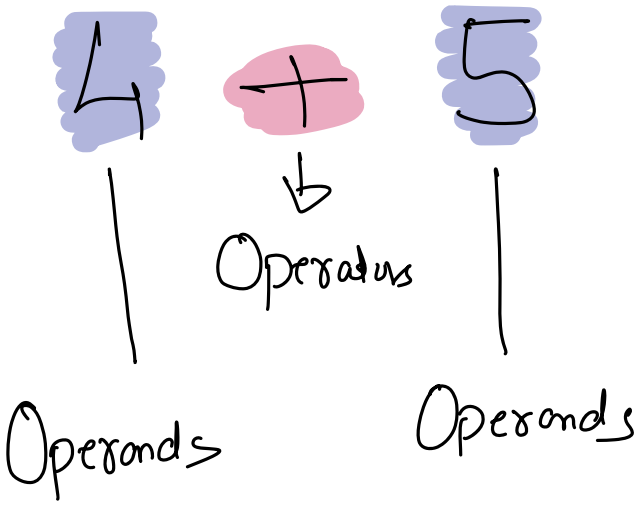
⇒ Ka. reverse a K

Final Stack

TC : O(n)

SC : O(n)

Infix Expressions → Operators come between Operands



Postfix Expression : Operands come before operators.

	Infix		Postfix
1)	4 + 5	⇒	4 5 +
2)	a + b x c a + bc x	⇒	a b c x +
3)	10 + 3x4 - 7 10 + 34x - 7 10 34x + - 7	⇒	10 34 * + 7 -

* = x

⇒ ( ) , / x , + −

⇒ If you have same priority, go left-right

(10 + 3) * 2 − (7 / 6) * (4 + 8)

10 3 + * 2 − 7 6 / * 4 8 +

10 3 + 2 * − 7 6 / * 4 8 +

10 3 + 2 * 7 6 / * 4 8 +

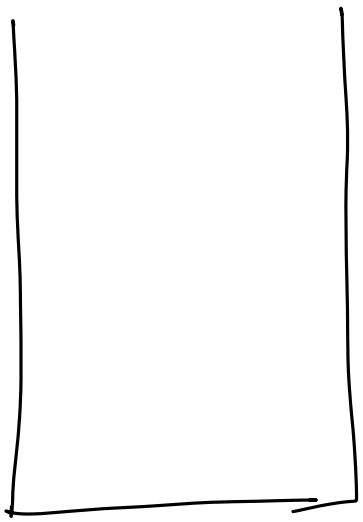
10 3 + 2 * 7 6 / * 4 8 + * −

$10/(4-2) \times 6 + 9 \Rightarrow 39$

$\frac{10}{42-} \times 6 + 9$

$10\ 42-/ \times 6 + 9$

$10\ 42-/6\times + 9$



$10\ 42-/6\times 9 +$

39

Infix Notation → (Homework) → Postfix Notation → (Assignment) → evaluate