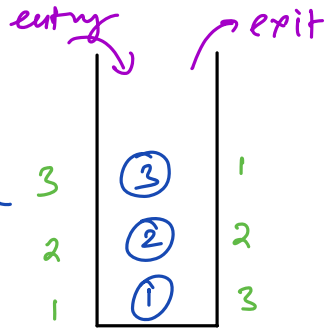
# Stacks 1 : Implementation & Basic Problems

→ linear data structure

→ data inserted from bottom to top

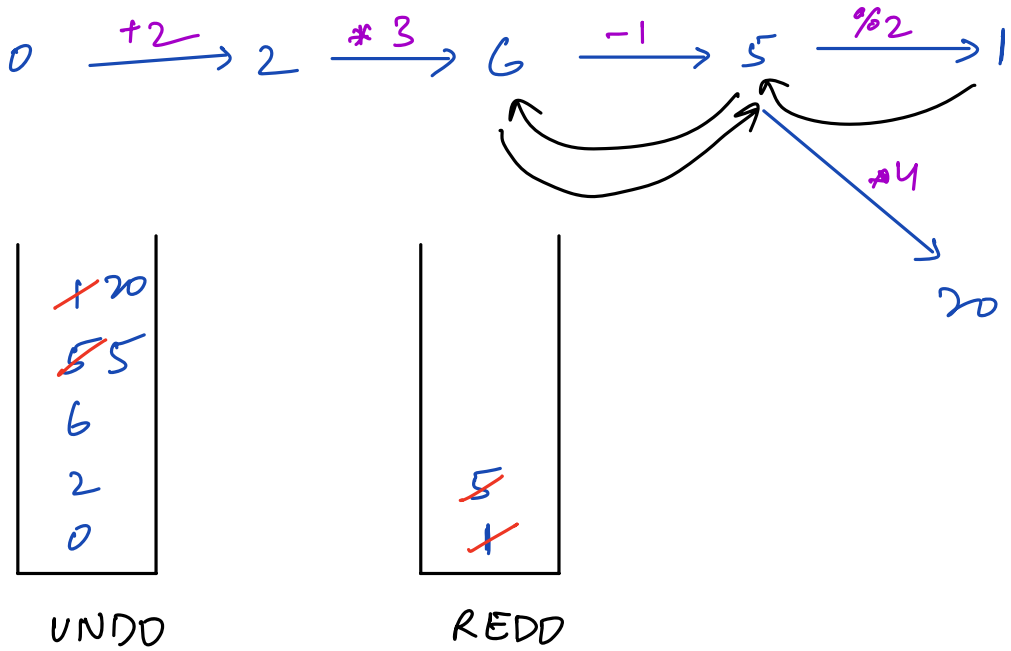
data accessed/deleted from top to bottom

⇒ Last In first Out (LIFO)




Examples

1. Pile of Plates 
2. Stack of chairs
3. Recursion
4. UNDO / REDO



1. Perform operation → insert in UNDO stack + empty the REDO stack

2. Perform UNDO → move from UNDO to REDO stack

3. Perform REDO → move from REDO to UNDO stack

## Operations of Stack

1. push(x) → insert element x on top of stack
2. pop() → remove top element of stack
3. peek() / top() → get the top element of stack
4. isEmpty() → check if stack is empty
5. clear(), size(), ....

} TC = O(1)

# Implement Stack using Arrays

push(1)

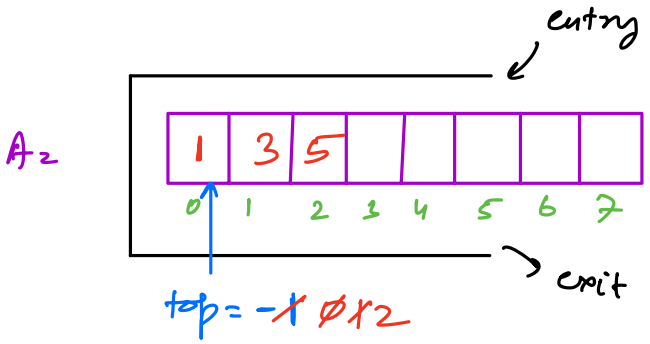
push(3)

push(5)

isEmpty() → false

pop()

peek() → 5




// stack → index 0 to top

```
void push(int x) {
    top++
    A[top] = x
}

bool isEmpty() {
    return (top == -1);
}

int pop() {
    if(isEmpty())  // Underflow
        return -1
    top--
    return A[top+1]  // return the deleted element
}
```

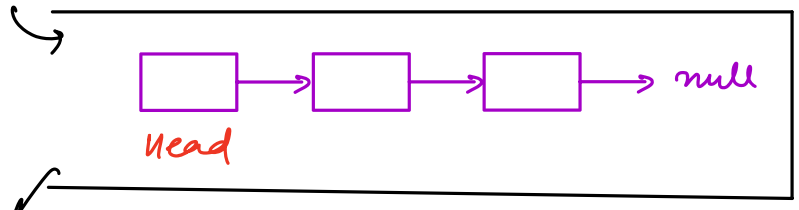
```
int peek() {
    if(isEmpty())
        return -1
    return A[top]
}
```

## Overflow

1. Restrict insertion over the defined size of array
2. Use dynamic Array

TC = O(1) for all operations

# Implement stack using linked list




push(x) → insert at Head

pop() → remove Head node

peek() → return Head.data → handle underflow

isEmpty() → (Head == NULL)

TC = O(1) for all operations

## Question

Check if the given sequence of parantheses containing only → { } ( ) [ ] is valid.

Valid → 1. count of '{' = count of '}'

count of '(' = count of ')'

count of '[' = count of ']'

2. Travelling from left to right # open >= # close

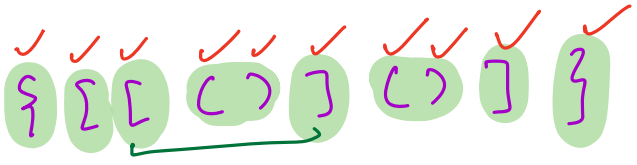
3. if opening of one type, there closing of

the same type in order w/o overlapping with another type

{ [ ) } ( ] → not valid

( ( ) { } ) → valid

{ ( [ ) ] } → not valid



check latest unpaired open bracket

LIFO ⇒ stack

Code

```
Stack<char> st

for( i=0 to n-1) {
    if(a[i] == '(' || a[i] == '[' || a[i] == '{') {
        st.push(a[i])
    }
```

```
else {
    if (st.empty()) return false        → invalid
    ch = st.pop()
    if (a[i] == ')' && ch != '(') return false
    if (a[i] == ']' && ch != '[') return false
    if (a[i] == '}' && ch != '{') return false
}
}
return true → st.isEmpty()
```

{ [ ] ( )

## Question

Given a string, remove equal pair of consecutive elements multiple times till it is possible.

Return final string as answer.

eg a b ~~c c~~ d → a b d

a ~~b b~~ ~~c c~~ d → a d

a b ~~c c~~ b → a ~~b b~~ → a

a ~~b b~~ c b c ~~d d~~ c → a c b ~~c c~~ → a c b

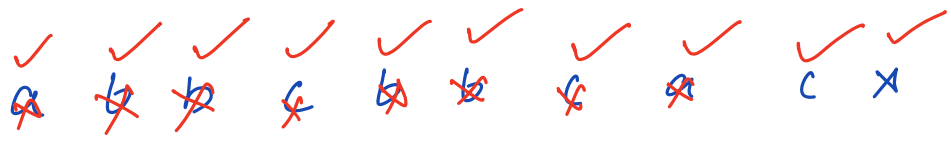
a ~~b b~~ c ~~b b~~ c a c x

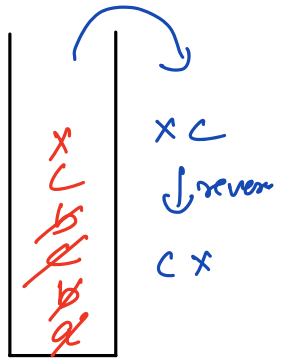
a ~~c c~~ a c x

~~a a~~ c x

c x

Store from left to right
remove right to left
} LIFO → stack






Code

```
Stack<char> st

for(i=0 to n-1) {
    if( !st.isEmpty() && st.peek() == a[i]) {
        st.pop()
    }
    else {
        st.push(a[i])
    }
}

String ans = ""

while(!st.isEmpty()) {
    ans += st.pop()
}

reverse(ans)
```

T.C = O(N)

S.C = O(N)

return ans

"abcd".append('e') ⇒ "abcde"

O(1)

"abcd" + "e" ⇒ "abcde"

O(size of string)

'a' + "bcde" ⇒ "abcde"

O(size of string)

Question

| Infix | Postfix |
|---|---|
| 2+5 | 2 5 + |
| operand1 operator operand2 | operand1 operand2 operator |

((6-5)*2)

((65-)*2) ⇒ 65- 2*

6-5=1

1*2=2

(6-(5*2))

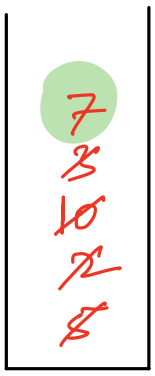
(6-(52*)) ⇒ 6 52* -

52* → 10

6-10=-4

Given a postfix expression, evaluate it.

[5, 2, *, 3, -]




[3, 5, +, 2, -, 2, 5, *, -]


Code

```
Stack<int> st

for ( i=0 to n-1 ) {
    if ( a[i] is operand / integer ) {
        st.push(a[i])
    }
    else {
```

```
        operator = getOperator(a[i])
        y = st.pop()
        x = st.pop()
        z = x operator y
        st.push(z)
    }
}
return st.peek()
```

pseudocode

write proper code in your language

TC = O(N)

SC = O(N)

## DOUBT