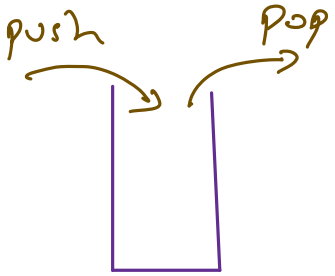
Welcome ☺

Agenda: Stacks
Implementation Arrays / LL
Opera^n's
2 ques^n's

Introduction

1) Glass of water
2) Pile of plates
3) Recursion

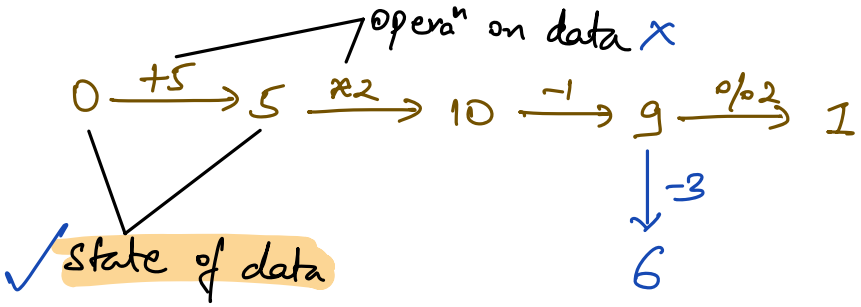
} LIFO (Last In First)

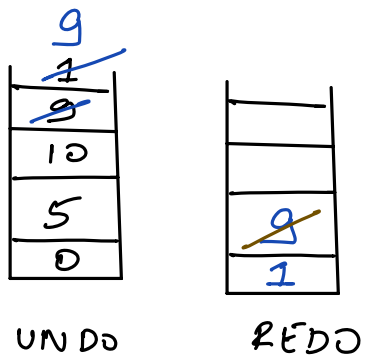


⇒ A stack is a linear D.S that stores information in sequence from bottom to top

⇒ It follows LIFO.

4) Calculator. ⇒ UNDO / REDO



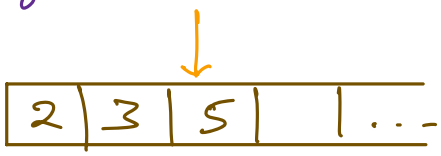


→ empty REDO stack if any new opera^n is performed.

Operations on Stack

1) Push(x) → insert x at top of stack.
2) Pop() → remove top element from stack
3) Peek/Top() → gets the topmost element from stack
4) IsEmpty() → checks if stack is empty.

Q Implement stack using arrays.



push(2)
push(3)
push(5)
pop() ⟶
peek() ⟶ 3
isEmpty() ⟶ false
push(10)

0 ⟶ t

```
t = -1
void push(x)
{
    t++
    A[t] = x
}
```

```
void pop()
{
    if(!isEmpty())
        t--
}
```

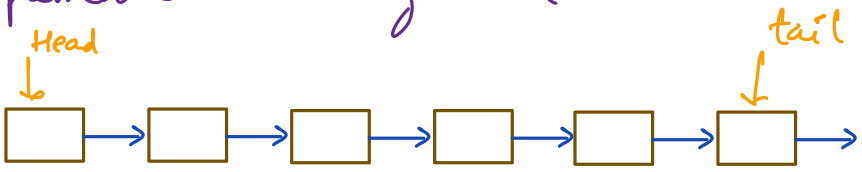
```
int peek()
{
    if(!isEmpty())
        return A[t]
    return -1
}
```

```
bool isEmpty()
{
    return t == -1
}
```

1) Overflow
→ use dynamic arrays.

2) Underflow

Q Stacks implementation using LL.



1) push → insert at head
2) pop → remove from head
3) peek → return Head.data.
4) isEmpty() → return (Head == NULL)

∀ T.C ⇒ O(1)

Q Balanced Paranthesis

Check whether the given sequence of paranthesis is valid / invalid.

( )

eg: ( ) ( ) (( )) ✓

(( ) (( )) ✗

→ 1) ∀ closing bracket → the last opening bracket should match
2) ∀ opening bracket → there should be a corresponding closing bracket.

→ while travelling L → R
1) opening '(' ≥ ')' at all indexes.
2) Total# open '(' == Total# close ')'

Code

```
open = 0      close = 0
for( i -> 0 to N-1)
{
  if( A[i] == '(' )
      open ++
  else
      close ++
  if( close > open ) return false.
}
if( open == close) return True
else return false.
```

T.C O(N)

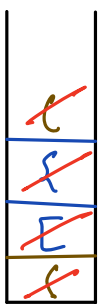
S.C O(1)

eg: ( ) [{ } ( )]

( ) ⇒ ( ) [( )] { } ✓

{ } ⇒ ( ) [( ))] ✗ (x)

[ ] ⇒ { ([ ) }] ✗



⇒ ∀ closing bracket → check if it is matching the last open bracket

Code

```
for( i -> 0 to N-1)
{ if( A[i] == '(' || A[i] == '{' || A[i] == '[')
      st.push(S[i])
  elseif( A[i] == ')')
  {
    if( st.empty() || st.pop() != '(' ) return false
  }
  elseif( A[i] == '}')
  {
    if( st.empty() || st.pop() != '{' ) return false
  }
  elseif( A[i] == ']')
  {
    if( st.empty() || st.pop() != '[' ) return false
  }
}
return st.Isempty()
```

T.C ⇒ O(N)

S.C ⇒ O(N)

Q Given a string S, remove equal pair of consecutive elements multiples times till possible & return the final string.

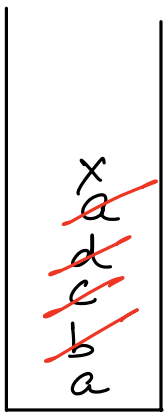
eg: abbc ⇒ ac

abccbde ⇒ abbde ⇒ ade

abbbe ⇒ abe

⇒ last element travelled should be removed first.

eg: abcddcaabx



H.W

T.C ⇒ O(N)

S.C ⇒ O(N)

Infix Expression

a + b

a * b - c

a * (b - c)

Op1 operator Op2

Postfix Expression

a b +

ab * c -

abc - *

Op1 Op2 Operator

eg: 10 6 - ⇒ 4

3 5 + 2 - 2 5 * -

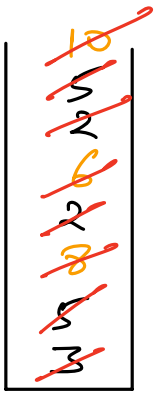
8

6

10

6 - 10 ⇒ -4

3 5 + 2 − 2 5 * −



3 + 5 = 8

8 − 2 = 6

2 * 5 = 10

6 − 10 = −4

Ans

operand → push it in the stack

operator → pop last two values/operands from the stack and perform operation
And push the resultant value back to stack.

op2 → pop()

op1 → pop()

push (op1 operator op2)

T.C O(N)

S.C O(N)