Stacks & Queues

① stack using array

1- in a class stack (Private)

- define an array size - arr[100]
- keep a variable to track index of top element [Top]
- a variable to assign max capacity.

1.1- Public constructor

- initialize capacity to take in user input size
- initialize top = -1.

1.2 Member functions

- bool is Empty ()

  if top = -1 then return -1

- bool is full ()

  if top = capacity -1
  then return full

- void push (int x)

  if stack isnt full
  then move top ahead
  and insert element.

__/__/____

- void pop()
  if stack isnt empty
  then top --
  now return arr[top]

- void peek()
  if Stack isnt empty
  return top element

- void display()
  - if stack empty return empty
  - else iterate from 0 -> top
  & print eles.

_/_/_

Q. Queue using Arrays

- in enqueue & dequeue
just remember that
you could just do rear++ or front ++
to move the pointers

- but if front or rear is at last &
you have to enqueue something in rear which is
at last but empty space is at start then
you do rear = (rear+1) % capacity that will
~~give~~ bring back rear to start.

- & after inserting if front is at last you have
to bring it back then something again.

__/__/____

Stacks & queues

⑤ Balanced Poranthesis.

- just iterate through the given String
- if current char is an opening bracket just push it to stak.
- if the current char is a closing bracket then look ink the stack if the top matches if it doesn't you can return unbalanced directly.
- but if it does match keep going till the end matching the closing ones with the opening ones until the stack is empty & then return true (Balanced).

Monotonic stack

① Next greater element
(a simple array)

# OP - iterate through the array backwards
- for each element check if ~~> stack is~~
  └ stack is $\overset{not}{}$empty & st.top <= currele
  then keep popping till you find a greater ele than curr
- now check if stack is empty if yes then store -1 in ans vector
- else store st.top in the ans vector
- then just push the current ele to stack
- finaly return ans vector.

(when youre given 2 arrays)
where arr1 is a subset of arr2.

# - just iterate through the arr2 from back
- for every element checkstack for greater element (if smaller pop till you find greater or becomes empty)
- now if stack is empty NGE for current number is -1 so in map store {curr : -1}
- if you find a greater element than current then in map store {curr : st.top()}
  └> NGE

Now the map would have all NGEs of Arr2. {arr2ele : NGE}

- Now just iterate through arr1 & for each element access NGE in map and push it to ans vector
- now return ans.

(when given a circular array)

# - basically iterating from 2n-1 → 0 Hypothetically extends the array

- so till i>n i.e in hypothetically part we just store all arr elements into stack

- now in actual iteration i<n we look for NGEs & store in ans usual way.

Ⓠ Min stack

- just declare a stack that stores a pair {curr, min}

- while pushing into a stack just if its empty then push {curr, $\underbrace{curr}_{\text{initial min}}$}

if not empty then {curr, min(curr, st.top().second)}

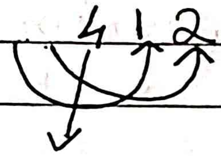
⑧ Stack using Queue

- basically you declare a queue & that should behave as a stack
- so ~~to~~ just while pushing
  - get the size of the current queue
  - push a value into queue
  - now loop through the size and basically reverse the queue

ex: q.size = 2   front [1, 2]

i = 0→1

push(4) -



newfront
i.e the
last pushed ele [LIFO]

⑨ Queue using stack

- basicaly you declare ~~a~~ 2 stack & that should work like a queue
- 1 stack takes in new eles 2nd stack is reverse of ~~the~~ 1st stack
- while pushing directly push in the stack 1
- while popping or getting top then push all elements from s1 to s2 & then pop or top.