1. Immediate Smaller Element

Given an integer array Arr of size N. For each element in the array, check whether the right adjacent element (on the next immediate position) of the array is smaller. If next element is smaller, update the current index to that element. If not, then -1.

Example 1:

Input:
N = 5
Arr[] = {4, 2, 1, 5, 3}
Output:
2 1 -1 3 -1
Explanation: Array elements are 4, 2, 1, 5 3. Next to 4 is 2 which is smaller, so we print 2. Next of 2 is 1 which is smaller, so we print 1. Next of 1 is 5 which is greater, so we print -1. Next of 5 is 3 which is smaller, so we print 3. Note that for last element, output is always going to be -1 because there is no element on right.
Example 2:

Input:
N = 6
Arr[] = {5, 6, 2, 3, 1, 7}
Output:
-1 2 -1 1 -1 -1
Explanation: Next to 5 is 6 which is greater, so we print -1.Next of 6 is 2 which is smaller, so we print 2. Next of 2 is 3 which is greater, so we print -1. Next of 3 is 1 which is smaller, so we print 1. Next of 1 is 7 which is greater, so we print -1. Note that for last element, output is always going to be -1 because there is

no element on right.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(1)

Constraints:
$1 \leq N \leq 107$
$1 \leq Arr[i] \leq 105$

2. Implement stack using array

Write a program to implement a Stack using Array. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack.

Example 1:

Input:
push(2)
push(3)
pop()
push(4)
pop()
Output: 3, 4
Explanation:
push(2) the stack will be {2}
push(3) the stack will be {2 3}
pop() poped element will be 3,
the stack will be {2}
push(4) the stack will be {2 4}
pop() poped element will be 4
Example 2:

Input:
pop()

push(4)
push(5)
pop()
Output: -1, 5

Expected Time Complexity : O(1) for both push() and pop().
Expected Auixilliary Space : O(1) for both push() and pop().

Constraints:
1 <= Q <= 100
1 <= x <= 100

3. Implement two stacks in an array

Your task is to implement 2 stacks in one array efficiently. You need to implement 4 methods.
push1 : pushes element into first stack.
push2 : pushes element into second stack.
pop1 : pops element from first stack and returns the popped element. If first stack is empty, it should return -1.
pop2 : pops element from second stack and returns the popped element. If second stack is empty, it should return -1.

Example 1:

Input:
push1(2)
push1(3)
push2(4)
pop1()
pop2()
pop2()
Output:
3 4 -1
Explanation:

push1(2) the stack1 will be {2}
push1(3) the stack1 will be {2,3}
push2(4) the stack2 will be {4}
pop1() the poped element will be 3 from stack1 and stack1 will be {2}
pop2() the poped element will be 4 from stack2 and now stack2 is empty
pop2() the stack2 is now empty hence returned -1.
Example 2:

Input:
push1(1)
push2(2)
pop1()
push1(3)
pop1()
pop1()
Output:
1 3 -1
Explanation:
push1(1) the stack1 will be {1}
push2(2) the stack2 will be {2}
pop1() the poped element will be 1 from stack1 and stack1 will be empty
push1(3) the stack1 will be {3}
pop1() the poped element will be 3 from stack1 and stack1 will be empty
pop1() the stack1 is now empty hence returned -1.

Expected Time Complexity: O(1) for all the four methods.
Expected Auxiliary Space: O(1) for all the four methods.

Constraints:
1 <= Number of queries <= 104
1 <= Number of elements in the stack <= 100
The sum of count of elements in both the stacks < size of the given array

4. Delete middle element of a stack

Given a stack, delete the middle element of the stack without using any additional data structure.
Middle element:- floor((size_of_stack+1)/2) (1-based indexing) from bottom of the stack.

Note: The output shown by the compiler is the stack from top to bottom.

Example 1:

Input:
Stack = {10, 20, 30, 40, 50}
Output:
ModifiedStack = {10, 20, 40, 50}
Explanation:
If you print the stack the bottom-most element will be 10 and the top-most element will be 50. Middle element will be element at index 3 from bottom, which is 30. Deleting 30, stack will look like {10 20 40 50}.
Example 2:

Input:
Stack = {10 20 30 40}
Output:
ModifiedStack = {10 30 40}
Explanation:
If you print the stack the bottom-most element will be 10 and the top-most element will be 40. Middle element will be element at index 2 from bottom, which is 20. Deleting 20, stack will look like {10 30 40}.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(N)

Constraints:
$2 \leq$ size of stack ≤ 105

5. Implement Stack using Linked List

Let's give it a try! You have a linked list and you have to implement the functionalities push and pop of stack using this given linked list. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack.

Example 1:

Input:
push(2)
push(3)
pop()
push(4)
pop()
Output: 3 4
Explanation:
push(2) the stack will be {2}
push(3) the stack will be {2 3}
pop() poped element will be 3, the stack will be {2}
push(4) the stack will be {2 4}
pop() poped element will be 4
Example 2:

Input:
pop()
push(4)
push(5)
pop()
Output: -1 5

Expected Time Complexity: O(1) for both push() and pop().
Expected Auxiliary Space: O(1) for both push() and pop().

Constraints:
1 <= Q <= 100
1 <= x <= 100

6. Reverse a string using Stack

You are given a string S, the task is to reverse the string using stack.

Example 1:

Input: S="GeeksforGeeks"

Output: skeeGrofskeeG

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(N)

Constraints:
$1 \leq$ length of the string ≤ 100

7. Special Stack

Design a data-structure SpecialStack that supports all the stack operations like push(), pop(), isEmpty(), isFull() and an additional operation getMin() which should return minimum element from the SpecialStack. Your task is to complete all the functions, using stack data-Structure.

Example 1:

Input:
Stack: 18 19 29 15 16
Output: 15
Explanation:
The minimum element of the stack is 15.

Note: The output of the code will be the value returned by getMin() function.

Expected Time Complexity: O(N) for getMin, O(1) for remaining all 4 functions.
Expected Auxiliary Space: O(1) for all the 5 functions.

Constraints:
$1 \leq N \leq 104$

8. Evaluation of Postfix Expression

Given string S representing a postfix expression, the task is to evaluate the expression and find the final value. Operators will only include the basic arithmetic operators like *, /, + and -.

Example 1:

Input: S = "231*+9-"
Output: -4
Explanation:
After solving the given expression,
we have -4 as a result.
Example 2:

Input: S = "123+*8-"
Output: -3
Explanation:
After solving the given postfix
expression, we have -3 as a result.

Expected Time Complexity: O(|S|)
Expected Auxiliary Space: O(|S|)

Constraints:
$1 \leq |S| \leq 105$

$0 \leq |Si| \leq 9$ (And given operators)

9. Get min at pop

Now, we'll try to solve a famous stack problem.
You are given an array A of size N. You need to first push the elements of the array into a stack and then print minimum in the stack at each pop until stack becomes empty.

Example 1:

Input:

N = 5
A = {1 2 3 4 5}
Output:
1 1 1 1 1
Explanation:
After pushing elements to the stack,
the stack will be "top -> 5, 4, 3, 2, 1".
Now, start popping elements from the stack
popping 5: min in the stack is 1.popped 5
popping 4: min in the stack is 1. popped 4
popping 3: min in the stack is 1. popped 3
popping 2: min in the stack is 1. popped 2
popping 1: min in the stack is 1. popped 1
Example 2:

Input:
N = 7
A = {1 6 43 1 2 0 5}
Output:
0 0 1 1 1 1 1
Explanation:
After pushing the elements to the stack,
the stack will be 5->0->2->1->43->6->1.
Now, poping the elements from the stack:
popping 5: min in the stack is 0. popped 5
popping 0: min in the stack is 0. popped 0
popping 2: min in the stack is 1. popped 2
popping 1: min in the stack is 1. popped 1
popping 43: min in the stack is 1.
popped 43
popping 6: min in the stack is 1. popped 6
popping 1: min in the stack is 1. popped 1.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(N).

Constraints:
0 <= Ai <= 107

10. Make the array beautiful

Given an array of negative and non-negative integers. You have to make the array beautiful. An array is beautiful if two adjacent integers, arr[i] and arr[i+1] are either negative or non-negative. And you can do the following operation any number of times until the array becomes beautiful.

If two adjacent integers are different i.e. one of them is negative and other is non-negative, remove them.
Return the beautiful array after performing the above operation.

Note:An empty array is also a beautiful array. There can be many adjacent integers which are different as stated above. So remove different adjacent integers as described above from left to right.

Example 1:

Input: 4 2 -2 1
Output: 4 1
Explanation: As at indices 1 and 2 , 2 and -2 have different sign, they are removed. And the the final array is: 4 1.
Example 2:

Input: 2 -2 1 -1
Output: []
Explanation: As at indices 0 and 1, 2 and -2 have different sign, so they are removed. Now the array is 1 -1.Now 1 and -1 are also removed as they have different sign. So the final array is empty.

Expected Time Complexity: O(N)
Expected Space Complexity: O(N)

Constraints:
1 <= size of the array <= 105
-105 <= arr[i] <= 105

11. Reversing the equation

Given a mathematical equation that contains only numbers and +, -, *, /. Print the equation in reverse, such that the equation is reversed, but the numbers remain the same. It is guaranteed that the given equation is valid, and there are no leading zeros.

Example 1:

Input:
S = "20-3+5*2"
Output: 2*5+3-20
Explanation: The equation is reversed with numbers remaining the same.
Example 2:

Input:
S = "5+2*56-2/4"
Output: 4/2-56*2+5
Explanation: The equation is reversed with numbers remaining the same.

Expected Time Complexity: O(|S|).
Expected Auxiliary Space: O(|S|).

Constraints:
1<=|S|<=105
The string contains only the characters '0' - '9', '+', '-', '*', and '/'.

12. Stack designer

You are given an array arr of size N. You need to push the elements of the array into a stack and then print them while popping.

Example 1:

Input:
n = 5
arr = {1 2 3 4 5}
Output:
5 4 3 2 1
Example 2:

Input:
n = 7
arr = {1 6 43 1 2 0 5}
Output:
5 0 2 1 43 6 1

Constraints:
1 <= Ai <= 107

13. Stack Operations

This Java module deals with Stacks, Queues, and ArrayLists. We'll perform various operations on them.

Given a stack of integers and Q queries. The task is to perform operation on stack according to the query.

Note: use push() to add element in the stack, peek() to get topmost element without removal, pop() gives topmost element with removal, search() to return position if found else -1.

Input Format:

First line of input contains nubmer of testcases T. For each testcase, first line of input contains Q, number of queries. Next line contains Q queries seperated by space. Queries are as:

i x : (adds element x in the stack).

r : (returns and removes the topmost element from the stack).

h : Prints the topmost element.

f y : (check if the element y is present or not in the stack). Print "Yes" if present, else "No".

Output Format:
For each testcase, perform Q queries and print the output wherever required.

Constraints:
1 <= T <= 100
1 <= Q <= 103

Example:
Input:
2
6
i 2 i 4 i 3 i 5 h f 8
4
i 3 i 4 r f 3

Output:
5
No
Yes

Explanation:
Testcase 1: Inserting 2, 4, 3, and 5 onto the stack. Returning top element which is 5. Finding 8 will give No, as 8 is not in the stack.

14. Insert an Element at the Bottom of a Stack

You are given a stack St of N integers and an element X. You have to insert X at the bottom of the given stack.

Example 1:

Input:
N = 5
X = 2
St = {4,3,2,1,8}
Output:
{2,4,3,2,1,8}
Explanation:
After insertion of 2, the final stack will be {2,4,3,2,1,8}.

Example 2:

Input:
N = 3
X = 4
St = {5,3,1}
Output:
{4,5,3,1}
Explanation:
After insertion of 4, the final stack will be {4,5,3,1}.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(N)

Constraints:
1 <= N <= 105
1 <= X, Elements of Stack <= 103
Sum of N over all test cases doesn't exceeds 106

15. Sort a stack

Given a stack, the task is to sort it such that the top of the stack has the greatest element.

Example 1:

Input:
Stack: 3 2 1
Output: 3 2 1
Example 2:

Input:
Stack: 11 2 32 3 41
Output: 41 32 11 3 2

Expected Time Complexity: O(N*N)
Expected Auxilliary Space: O(N) recursive.

Constraints:
1<=N<=100