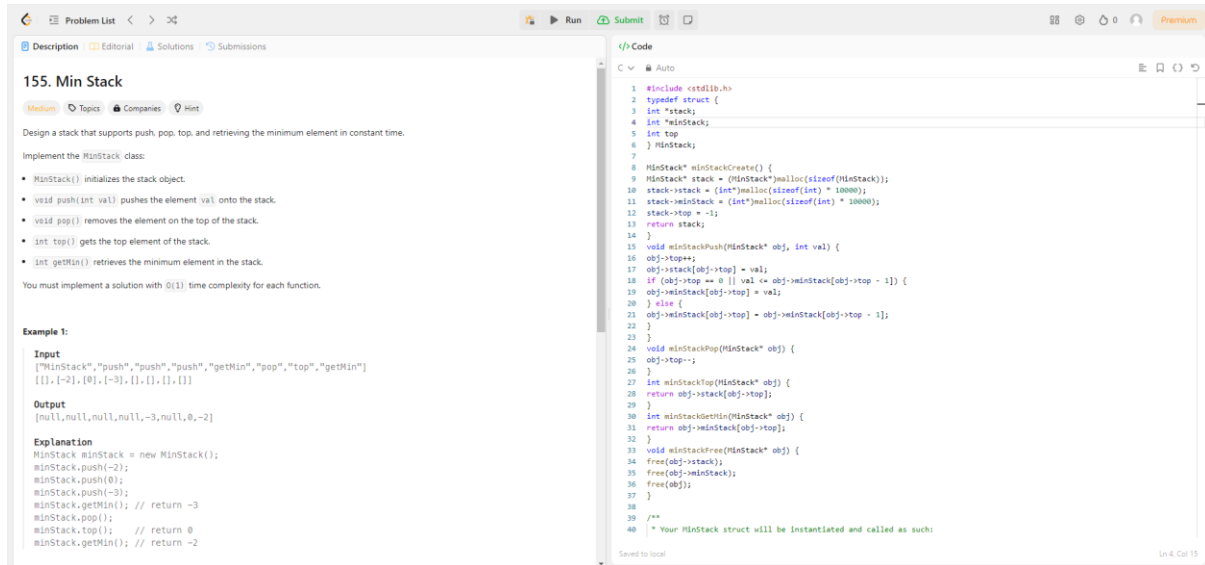


LEET CODE: Demonstration of stack program

SOURCE CODE:



The screenshot shows the LeetCode editor interface for the problem "155. Min Stack". The left pane contains the problem description, which asks for a stack that supports push, pop, top, and retrieving the minimum element in constant time. It includes an example input and output, and an explanation of the provided solution. The right pane shows the C++ code for the solution, which implements the MinStack class using a stack and a min variable to track the minimum element.

155. Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

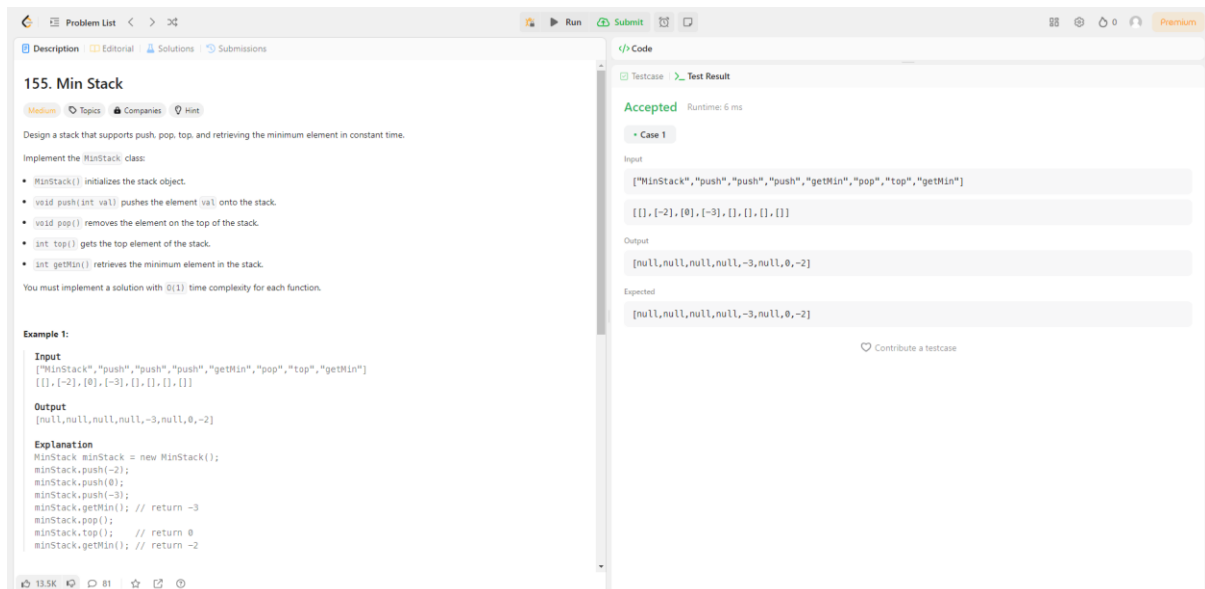
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

```
1 #include <stdlib.h>
2 typedef struct {
3     int *stack;
4     int *minStack;
5     int top;
6 } MinStack;
7
8 MinStack* minStackCreate() {
9     MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
10    stack->stack = (int*)malloc(sizeof(int) * 10000);
11    stack->minStack = (int*)malloc(sizeof(int) * 10000);
12    stack->top = -1;
13    return stack;
14}
15 void minStackPush(MinStack* obj, int val) {
16     obj->top++;
17     obj->stack[obj->top] = val;
18     if (obj->top == 0 || val <= obj->minStack[obj->top - 1]) {
19         obj->minStack[obj->top] = val;
20     } else {
21         obj->minStack[obj->top] = obj->minStack[obj->top - 1];
22     }
23 }
24 void minStackPop(MinStack* obj) {
25     obj->top--;
26 }
27 int minStackTop(MinStack* obj) {
28     return obj->stack[obj->top];
29 }
30 int minStackGetMin(MinStack* obj) {
31     return obj->minStack[obj->top];
32 }
33 void minStackFree(MinStack* obj) {
34     free(obj->stack);
35     free(obj->minStack);
36     free(obj);
37 }
38
39 /**
40  * Your MinStack struct will be instantiated and called as such:
41  */
```

OUTPUT:



The screenshot shows the LeetCode editor interface for the problem "155. Min Stack". The left pane contains the problem description and the C++ code. The right pane shows the test results, indicating that the solution is accepted and runs in 6 ms.

155. Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

Accepted Runtime: 6 ms

Case 1

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Expected
[null,null,null,null,-3,null,0,-2]

Contribute a test case