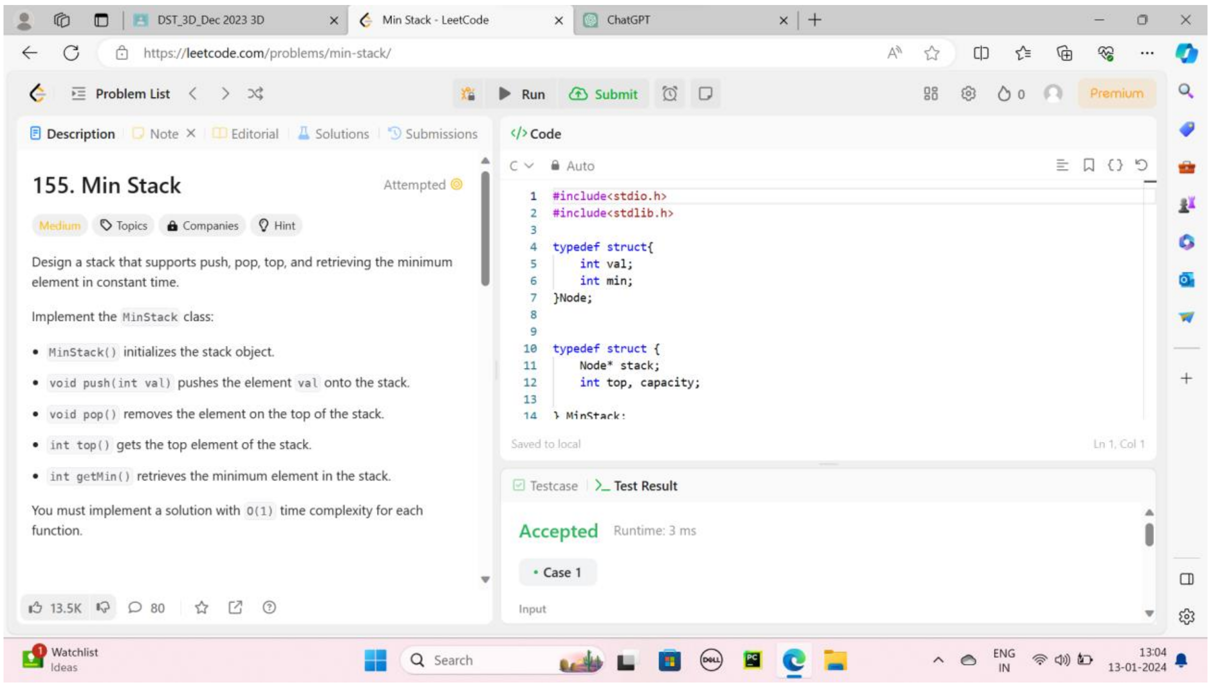


155. Min Stack

Attempted

Medium · Topics · Companies · Hint

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.

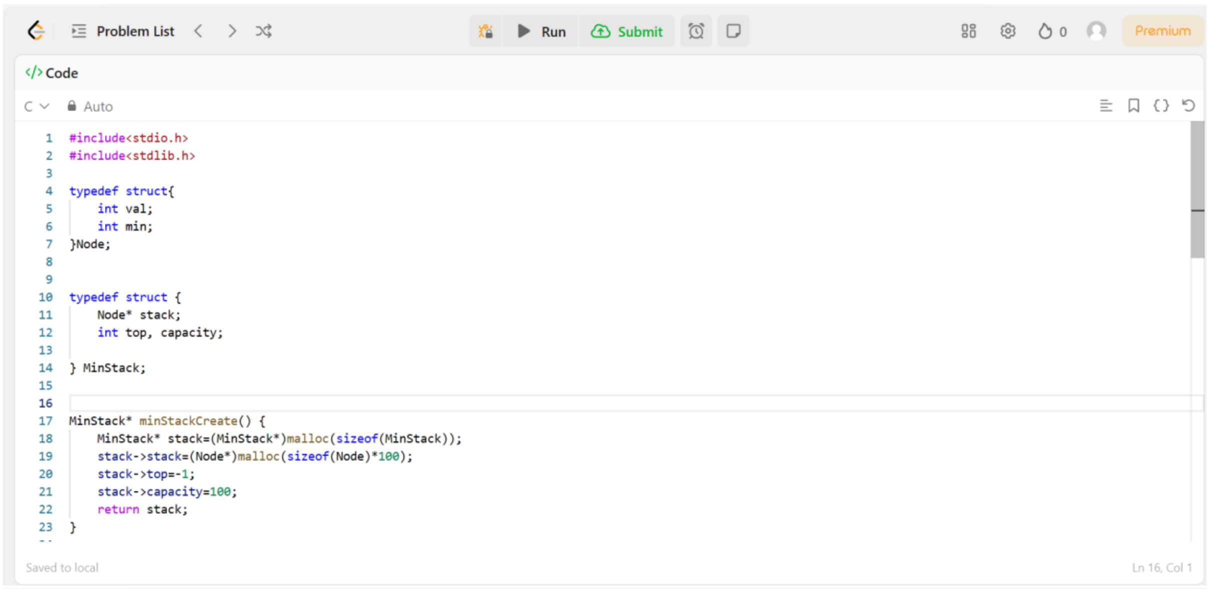
```c
#include<stdio.h>
#include<stdlib.h>

typedef struct{
    int val;
    int min;
}Node;


typedef struct {
    Node* stack;
    int top, capacity;

} MinStack;
```

Saved to local   Ln 1, Col 1

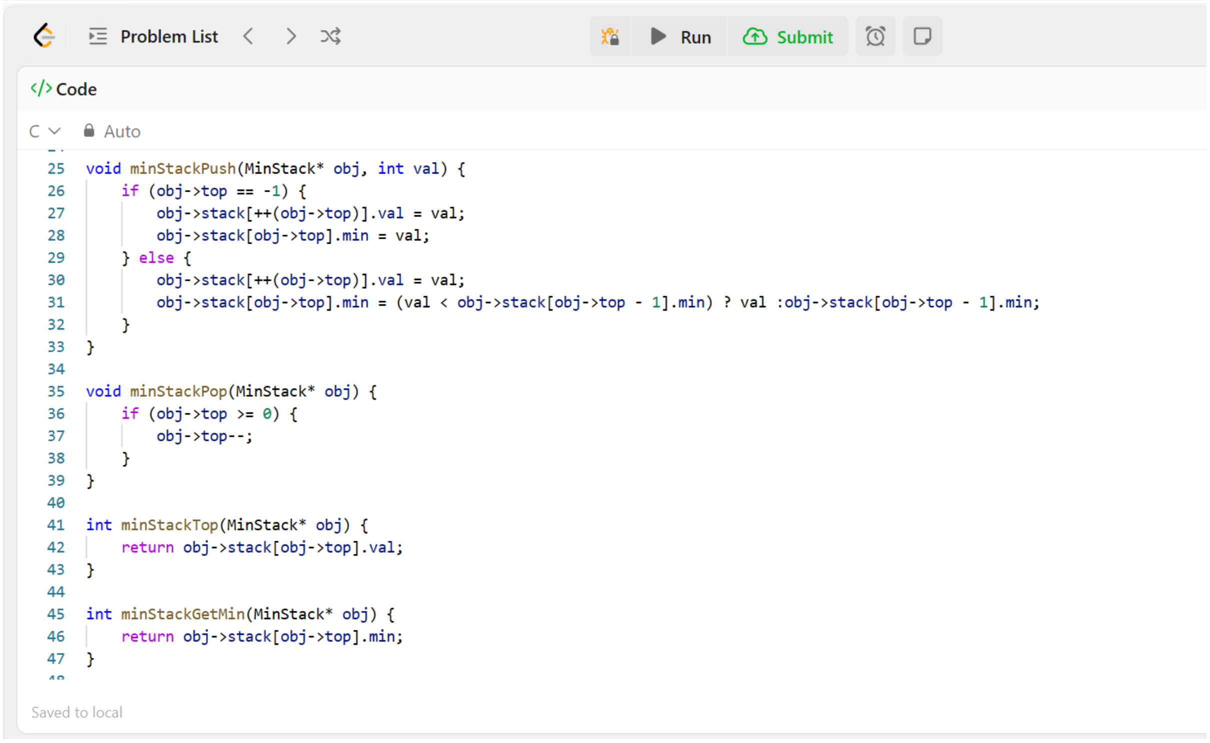
Accepted   Runtime: 3 ms

• Case 1

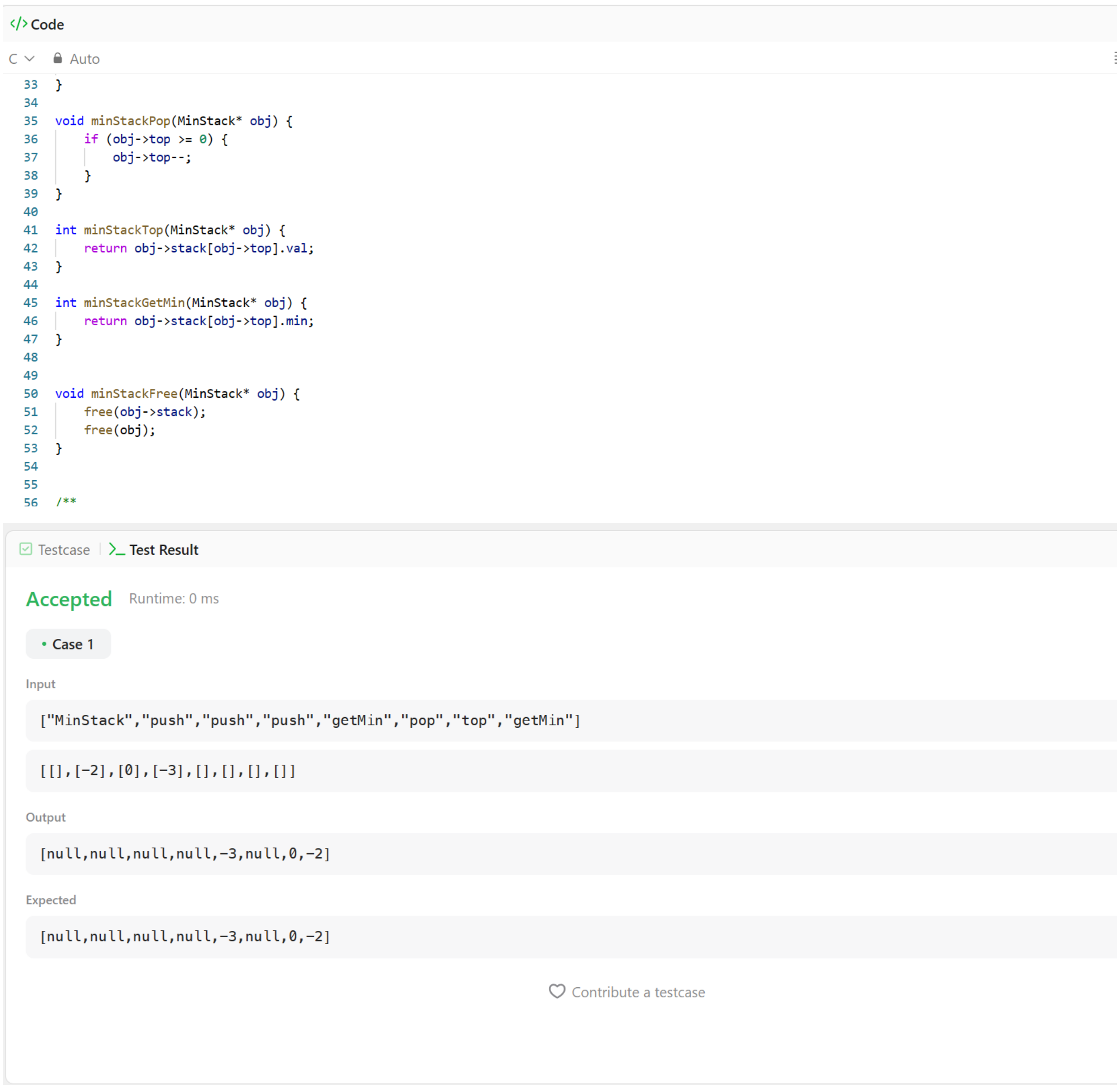
Input



```c
#include<stdio.h>
#include<stdlib.h>

typedef struct{
    int val;
    int min;
}Node;


typedef struct {
    Node* stack;
    int top, capacity;

} MinStack;


MinStack* minStackCreate() {
    MinStack* stack=(MinStack*)malloc(sizeof(MinStack));
    stack->stack=(Node*)malloc(sizeof(Node)*100);
    stack->top=-1;
    stack->capacity=100;
    return stack;
}
```



```c
void minStackPush(MinStack* obj, int val) {
    if (obj->top == -1) {
        obj->stack[++(obj->top)].val = val;
        obj->stack[obj->top].min = val;
    } else {
        obj->stack[++(obj->top)].val = val;
        obj->stack[obj->top].min = (val < obj->stack[obj->top - 1].min) ? val :obj->stack[obj->top - 1].min;
    }
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}

int minStackTop(MinStack* obj) {
    return obj->stack[obj->top].val;
}

int minStackGetMin(MinStack* obj) {
    return obj->stack[obj->top].min;
}
```


</> Code

C Auto

```c
}

void minStackPop(MinStack* obj) {
    if (obj->top >= 0) {
        obj->top--;
    }
}

int minStackTop(MinStack* obj) {
    return obj->stack[obj->top].val;
}

int minStackGetMin(MinStack* obj) {
    return obj->stack[obj->top].min;
}


void minStackFree(MinStack* obj) {
    free(obj->stack);
    free(obj);
}


/**
```

Testcase | >_ Test Result

Accepted Runtime: 0 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 testcase