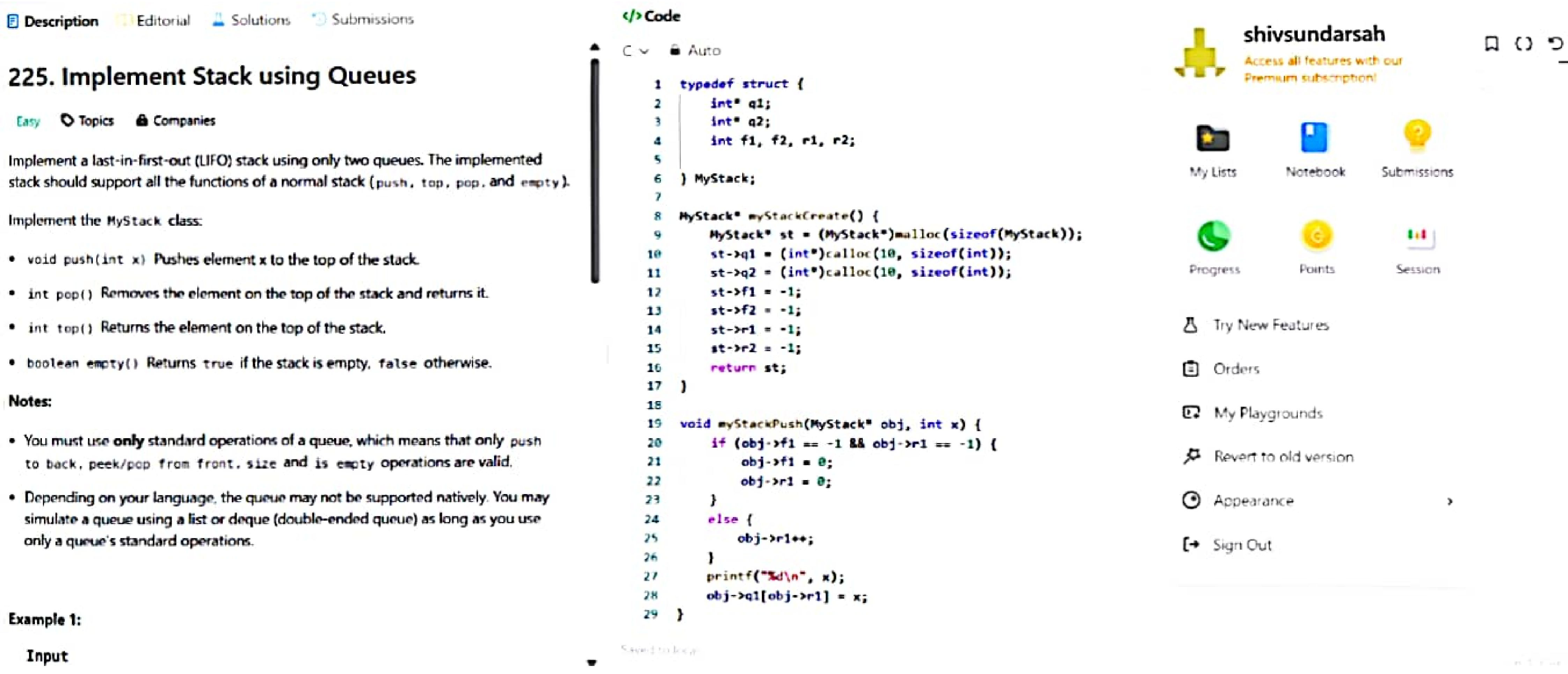


Description Editorial Solutions Submissions

225. Implement Stack using Queues

Easy Topics Companies

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

Input

</> Code

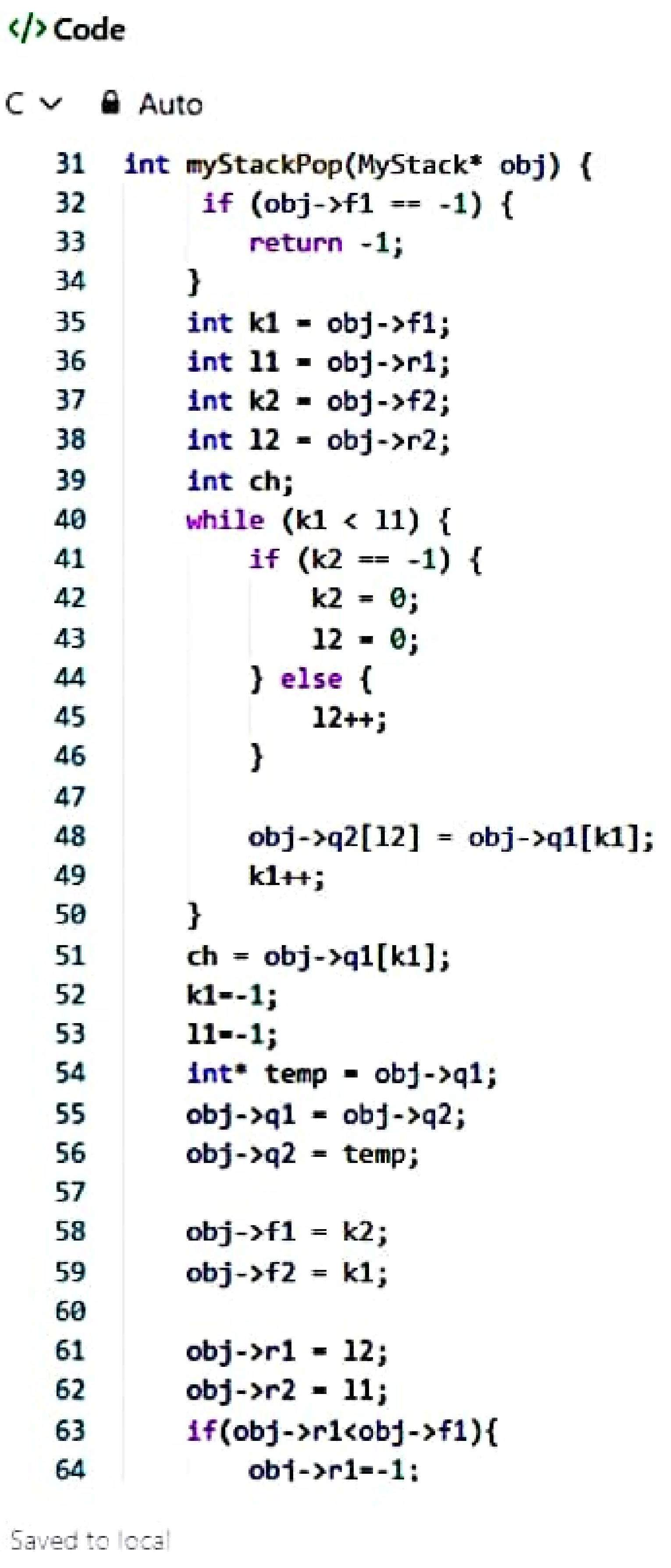
C Auto

```c
typedef struct {
    int* q1;
    int* q2;
    int f1, f2, r1, r2;

} MyStack;

MyStack* myStackCreate() {
    MyStack* st = (MyStack*)malloc(sizeof(MyStack));
    st->q1 = (int*)calloc(10, sizeof(int));
    st->q2 = (int*)calloc(10, sizeof(int));
    st->f1 = -1;
    st->f2 = -1;
    st->r1 = -1;
    st->r2 = -1;
    return st;
}

void myStackPush(MyStack* obj, int x) {
    if (obj->f1 == -1 && obj->r1 == -1) {
        obj->f1 = 0;
        obj->r1 = 0;
    }
    else {
        obj->r1++;
    }
    printf("%d\n", x);
    obj->q1[obj->r1] = x;
}
```

Saved to local

shivsundarsah

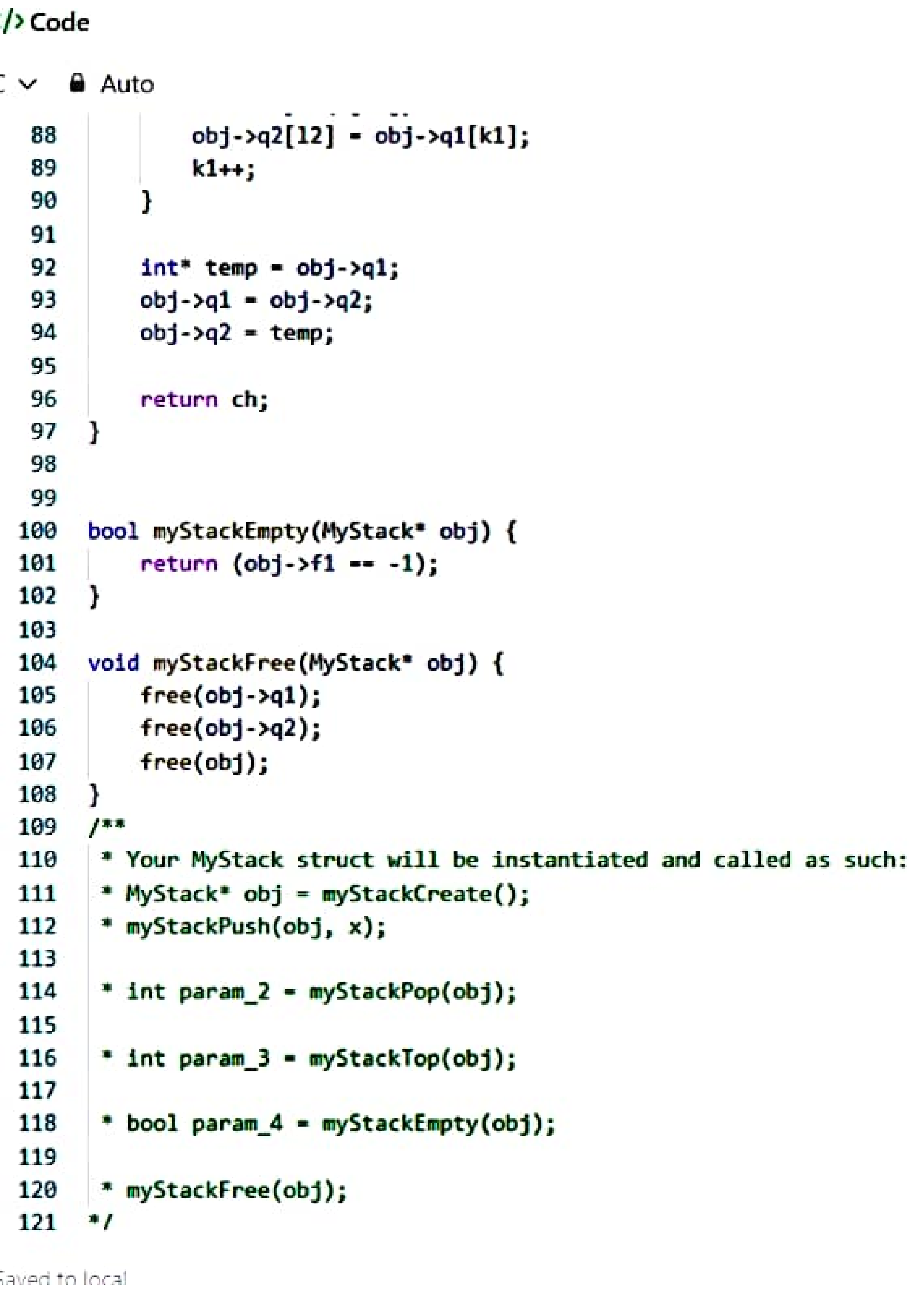
Access all features with our Premium subscription!


</> Code

C Auto

```c
int myStackPop(MyStack* obj) {
     if (obj->f1 == -1) {
        return -1;
    }
    int k1 = obj->f1;
    int l1 = obj->r1;
    int k2 = obj->f2;
    int l2 = obj->r2;
    int ch;
    while (k1 < l1) {
        if (k2 == -1) {
            k2 = 0;
            l2 = 0;
        } else {
            l2++;
        }

        obj->q2[l2] = obj->q1[k1];
        k1++;
    }
    ch = obj->q1[k1];
    k1=-1;
    l1=-1;
    int* temp = obj->q1;
    obj->q1 = obj->q2;
    obj->q2 = temp;

    obj->f1 = k2;
    obj->f2 = k1;

    obj->r1 = l2;
    obj->r2 = l1;
    if(obj->r1<obj->f1){
        obj->r1=-1;
```

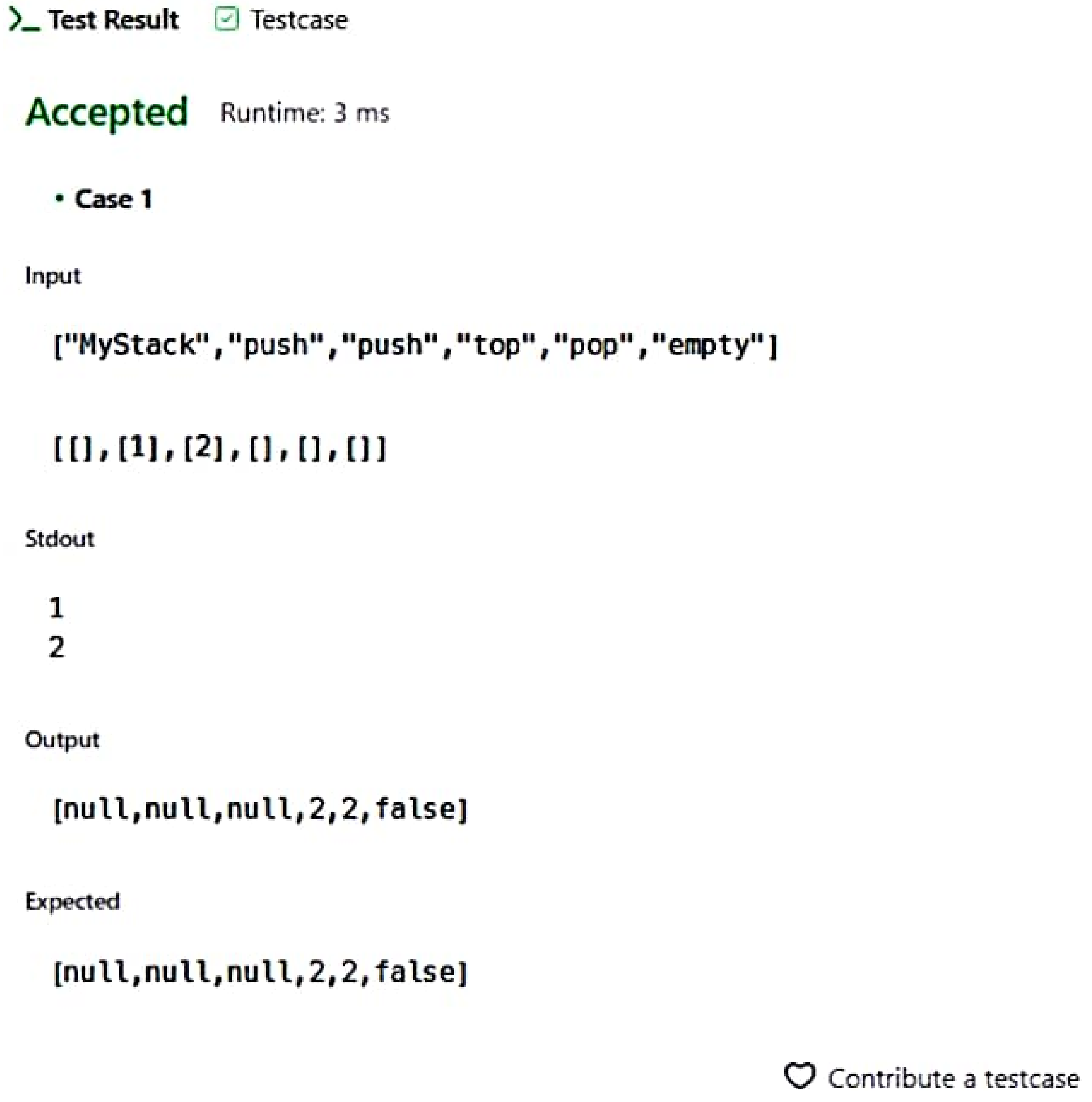
Saved to local


</> Code

C ⌄ 🔒 Auto

```c
            obj->q2[12] = obj->q1[k1];
            k1++;
        }

        int* temp = obj->q1;
        obj->q1 = obj->q2;
        obj->q2 = temp;

        return ch;
}


bool myStackEmpty(MyStack* obj) {
        return (obj->f1 == -1);
}

void myStackFree(MyStack* obj) {
        free(obj->q1);
        free(obj->q2);
        free(obj);
}
/**
 * Your MyStack struct will be instantiated and called as such:
 * MyStack* obj = myStackCreate();
 * myStackPush(obj, x);

 * int param_2 = myStackPop(obj);

 * int param_3 = myStackTop(obj);

 * bool param_4 = myStackEmpty(obj);

 * myStackFree(obj);
*/
```

Saved to local



>_ Test Result   ☑ Testcase

Accepted Runtime: 3 ms

- **Case 1**

Input

```
["MyStack","push","push","top","pop","empty"]

[[],[1],[2],[],[],[]]
```

Stdout

```
1
2
```

Output

```
[null,null,null,2,2,false]
```

Expected

```
[null,null,null,2,2,false]
```

♡ Contribute a testcase