225. Implement Stack using Queues

Easy 965 666 Add to List Share

Implement a last in first out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal queue (`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 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
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False
```

Constraints:

- `1 <= x <= 9`
- At most `100` calls will be made to `push` , `pop` , `top` , and `empty` .
- All the calls to `pop` and `top` are valid.

Follow-up: Can you implement the stack such that each operation is **amortized** `O(1)` time complexity? In other words, performing `n` operations will take overall `O(n)` time even if one of those operations may take longer. You can use more than two queues.

Push:
1. push element into queue 2.
2. pop all elements from queue 1, and push into queue 2.
3. exchange queue 1 and queue 2.

Time complexity: O(n)

```cpp
class MyStack {
public:
    /** Initialize your data structure here. */
    MyStack() {

    }

    /** Push element x onto stack. */
    void push(int x) {
        // push element into q2
        q2.push(x);

        // pop all elements from q1 and push into q2
        while (q1.size() != 0) {
            q2.push(q1.front());
            q1.pop();
        }

        // exchange q1 and q2
        queue<int> tmp = q1;
        q1 = q2;
        q2 = tmp;
    }

    /** Removes the element on top of the stack and returns that element. */
    int pop() {
        int x = q1.front();
        q1.pop();
        return x;
    }

    /** Get the top element. */
    int top() {
        return q1.front();
    }

    /** Returns whether the stack is empty. */
    bool empty() {
        return q1.empty();
    }

private:
    queue<int> q1;
    queue<int> q2;
};

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStack* obj = new MyStack();
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->top();
 * bool param_4 = obj->empty();
 */
```