Problem 232: Implement Queue using Stacks

Problem Information

Difficulty: Easy
Acceptance Rate: 68.81%
Paid Only: No
Tags: Stack, Design, Queue

Problem Description

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push`, `peek`, `pop`, and `empty`).

Implement the `MyQueue` class:

* `void push(int x)` Pushes element x to the back of the queue. * `int pop()` Removes the element from the front of the queue and returns it. * `int peek()` Returns the element at the front of the queue. * `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Notes:

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

Example 1:

Input ["MyQueue", "push", "push", "peek", "pop", "empty"] [[], [1], [2], [], [], []] **Output** [null, null, null, 1, 1, false] **Explanation** MyQueue myQueue = new MyQueue(); myQueue.push(1); // queue is: [1] myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue) myQueue.peek(); // return 1 myQueue.pop(); // return 1, queue is [2] myQueue.empty(); // return false

Constraints:

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

Follow-up: Can you implement the queue such that each operation is **[amortized](https://en.wikipedia.org/wiki/Amortized_analysis)** `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.

Code Snippets

C++:

```
class MyQueue {
public:
MyQueue() {

}

void push(int x) {

}

int pop() {

}

int peek() {

}

bool empty() {

}
};

/**
* Your MyQueue object will be instantiated and called as such:
* MyQueue* obj = new MyQueue();
* obj->push(x);
* int param_2 = obj->pop();
* int param_3 = obj->peek();
```

```
* bool param_4 = obj->empty();
*/
```

Java:

```
class MyQueue {

public MyQueue() {

}

public void push(int x) {

}

public int pop() {

}

public int peek() {

}

public boolean empty() {

}
}

/**
* Your MyQueue object will be instantiated and called as such:
* MyQueue obj = new MyQueue();
* obj.push(x);
* int param_2 = obj.pop();
* int param_3 = obj.peek();
* boolean param_4 = obj.empty();
*/
```

Python3:

```
class MyQueue:

def __init__(self):
```

```
def push(self, x: int) -> None:


def pop(self) -> int:


def peek(self) -> int:


def empty(self) -> bool:



# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```