Implement Queue using Stacks

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

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, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

Solution 1

I have one input stack, onto which I push the incoming elements, and one output stack, from which I peek/pop. I move elements from input stack to output stack when needed, i.e., when I need to peek/pop but the output stack is empty. When that happens, I move all elements from input to output stack, thereby reversing the order so it's the correct order for peek/pop.

The loop in `peek` does the moving from input to output stack. Each element only ever gets moved like that once, though, and only after we already spent time pushing it, so the overall amortized cost for each operation is O(1).

Ruby

```ruby
class Queue
    def initialize
        @in, @out = [], []
    end

    def push(x)
        @in << x
    end

    def pop
        peek
        @out.pop
    end

    def peek
        @out << @in.pop until @in.empty? if @out.empty?
        @out.last
    end

    def empty
        @in.empty? && @out.empty?
    end
end
```

Java

```
class MyQueue {

    Stack<Integer> input = new Stack();
    Stack<Integer> output = new Stack();

    public void push(int x) {
        input.push(x);
    }

    public void pop() {
        peek();
        output.pop();
    }

    public int peek() {
        if (output.empty())
            while (!input.empty())
                output.push(input.pop());
        return output.peek();
    }

    public boolean empty() {
        return input.empty() && output.empty();
    }
}
```

C++

```
class Queue {
    stack<int> input, output;
public:

    void push(int x) {
        input.push(x);
    }

    void pop(void) {
        peek();
        output.pop();
    }

    int peek(void) {
        if (output.empty())
            while (input.size())
                output.push(input.top()), input.pop();
        return output.top();
    }

    bool empty(void) {
        return input.empty() && output.empty();
    }
};
```

written by StefanPochmann original link here

Solution 2

```
class MyQueue {
Stack<Integer> queue = new Stack<Integer>();
// Push element x to the back of queue.
public void push(int x) {
    Stack<Integer> temp = new Stack<Integer>();
    while(!queue.empty()){
        temp.push(queue.pop());
    }
    queue.push(x);
    while(!temp.empty()){
        queue.push(temp.pop());
    }
}

// Removes the element from in front of queue.
public void pop() {
    queue.pop();
}

// Get the front element.
public int peek() {
    return queue.peek();
}

// Return whether the queue is empty.
public boolean empty() {
    return queue.empty();
}
```

}

written by yangneu2015 original link here

Solution 3

```
class MyQueue {
    Stack<Integer> pushStack = new Stack<>();
    Stack<Integer> popStack = new Stack<>();

    // Push element x to the back of queue.
    public void push(int x) {
        pushStack.push(x);
    }

    // Removes the element from in front of queue.
    public void pop() {
        if(popStack.isEmpty()) {
            while(!pushStack.isEmpty()) {
                popStack.push(pushStack.pop());
            }
        }
        popStack.pop();
    }

    // Get the front element.
    public int peek() {
        if(popStack.isEmpty()) {
            while(!pushStack.isEmpty()) {
                popStack.push(pushStack.pop());
            }
        }
        return popStack.peek();
    }

    // Return whether the queue is empty.
    public boolean empty() {
        return pushStack.isEmpty() && popStack.isEmpty();
    }
}
```

written by tuan.huu.minh.nguyen original link here