



Queues: A Tale of Two Stacks

by [saikiran9194](#)

Problem

Submissions

Leaderboard

Discussions

Editorial

Check out the resources on the page's right side to learn more about queues. The video tutorial is by Gayle Laakmann McDowell, author of the best-selling interview book [Cracking the Coding Interview](#).

A **queue** is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and new elements to be added to the rear. This is called a *First-In-First-Out* (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed.

A basic queue has the following operations:

- *Enqueue*: add a new element to the end of the queue.
- *Dequeue*: remove the element from the front of the queue and return it.

In this challenge, you must first implement a queue using *two stacks*. Then process q queries, where each query is one of the following **3** types:

1. x : Enqueue element x into the end of the queue.
2. Dequeue the element at the front of the queue.
3. Print the element at the front of the queue.

Input Format

The first line contains a single integer, q , denoting the number of queries.

Each line i of the q subsequent lines contains a single query in the form described in the problem statement above. All three queries start with an integer denoting the query *type*, but only query **1** is followed by an additional space-separated value, x , denoting the value to be enqueued.

Constraints

- $1 \leq q \leq 10^5$
- $1 \leq \text{type} \leq 3$
- $1 \leq |x| \leq 10^9$
- It is guaranteed that a valid answer always exists for each query of type **3**.

Output Format

For each query of type **3**, print the value of the element at the front of the queue on a new line.

Sample Input

```
10
1 42
2
1 14
3
1 28
3
1 60
1 78
```

2
2

Sample Output

14
14

Explanation

We perform the following sequence of actions:

1. Enqueue **42**; *queue* = {**42**}.
2. Dequeue the value at the head of the queue, **42**; *queue* = {}.
3. Enqueue **14**; *queue* = {**14**}.
4. Print the value at the head of the queue, **14**; *queue* = {**14**}.
5. Enqueue **28**; *queue* = {**14** ← **28**}.
6. Print the value at the head of the queue, **14**; *queue* = {**14** ← **28**}.
7. Enqueue **60**; *queue* = {**14** ← **28** ← **60**}.
8. Enqueue **78**; *queue* = {**14** ← **28** ← **60** ← **78**}.
9. Dequeue the value at the head of the queue, **14**; *queue* = {**28** ← **60** ← **78**}.
10. Dequeue the value at the head of the queue, **28**; *queue* = {**60** ← **78**}.

f t in

Submissions: 14250

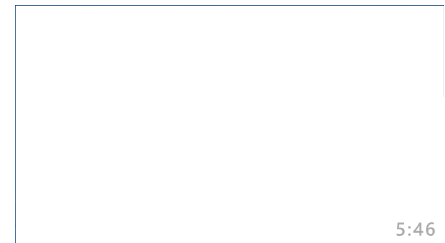
Max Score: 30

Difficulty: Medium

Rate This Challenge:

☆☆☆☆☆

Need Help?



[Queues](#)

[Stacks](#)

[More](#)

Current Buffer (saved locally, editable)  

Java 8



```
1 import java.io.*;
2 import java.util.*;
3 import java.text.*;
4 import java.math.*;
5 import java.util.regex.*;
6
7 class MyQueue {
8
```

```
9      Stack<Integer> s1 = new Stack<>();
10     Stack<Integer> s2 = new Stack<>();
11
12
13     //enqueue -- keep pushing onto s1
14     public void enqueue(int val) {
15
16         s1.push(val);
17     }
18
19     //dequeue -- remove front
20     public void dequeue () {
21
22         if (s2.isEmpty()){
23             prepStack();
24         }
25
26         s2.pop();
27     }
28
29     //print the front element
30     public int peek() {
31
32         if (s2.isEmpty())
33             prepStack();
34         return s2.peek();
35     }
36
37
38     //get all elements from s1 to s2 for front operations
39     public void prepStack() {
40         while (!s1.isEmpty()) {
41             int temp = s1.pop();
42             s2.push(temp);
43         }
44     }
45 }
46
47 public class solution {
48
49     public static void main(String[] args) {
50         MyQueue queue = new MyQueue();
51
52         Scanner scan = new Scanner(System.in);
53         int n = scan.nextInt();
54
55         for (int i = 0; i < n; i++) {
56             int operation = scan.nextInt();
57             if (operation == 1) { // enqueue
58                 queue.enqueue(scan.nextInt());
59             } else if (operation == 2) { // dequeue
60                 queue.dequeue();
61             } else if (operation == 3) { // print/peek
62                 System.out.println(queue.peek());
63             }
64         }
65         scan.close();
66     }
67 }
68
```

Line: 4 Col: 20

 [Upload Code as File](#)☐ Test against custom input[Run Code](#)[Submit Code](#)

Congrats, you solved this challenge!

✓ Test Case #0
✓ Test Case #3
✓ Test Case #6
✓ Test Case #9
✓ Test Case #12
✓ Test Case #15

✓ Test Case #1
✓ Test Case #4
✓ Test Case #7
✓ Test Case #10
✓ Test Case #13

✓ Test Case #2
✓ Test Case #5
✓ Test Case #8
✓ Test Case #11
✓ Test Case #14

[Next Challenge](#)

Copyright © 2017 HackerRank. All Rights Reserved

Join us on IRC at [#hackerrank](#) on freenode for hugs or bugs.

[Contest Calendar](#) | [Interview Prep](#) | [Blog](#) | [Scoring](#) | [Environment](#) | [FAQ](#) | [About Us](#) | [Support](#) | [Careers](#) | [Terms Of Service](#) | [Privacy Policy](#) | [Request a Feature](#)