Queues: A Tale of Two Stacks

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.

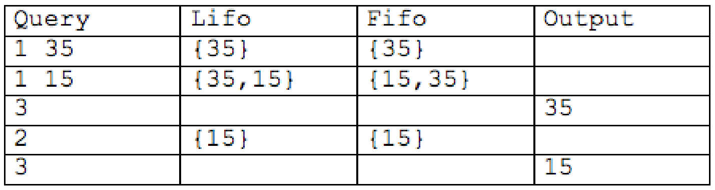
A basic queue has the following operations:

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

For example, a series of queries might be as follows:



Query	Lifo	Fifo	Output
1 35	{35}	{35}	
1 15	{35,15}	{15,35}	
3			35
2	{15}	{15}	
3			15

Function Description

Complete the *put*, *pop*, and *peek* methods in the editor below. They must perform the actions as described above.

Input Format

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

Each of the next q lines contains a single query in the form described in the problem statement above. All 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

- $1 \leq q \leq 10^5$
- $1 \leq type \leq 3$

- $1 \leq |x| \leq 10^9$
- It is guaranteed that a valid answer always exists for each query of types 2 and 3.

Output Format

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

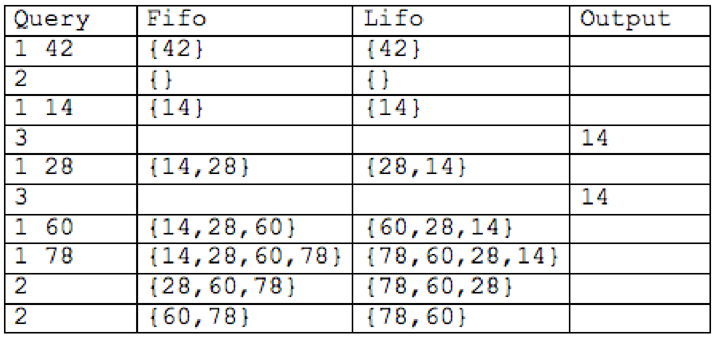
Sample Input

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

Sample Output

```
14
14
```

Explanation



Query	Fifo	Lifo	Output
1 42	{42}	{42}	
2	{}	{}	
1 14	{14}	{14}	
3			14
1 28	{14,28}	{28,14}	
3			14
1 60	{14,28,60}	{60,28,14}	
1 78	{14,28,60,78}	{78,60,28,14}	
2	{28,60,78}	{78,60,28}	
2	{60,78}	{78,60}	