

# Queue

## Problem ID: queue2

Write your own implementation of a queue, specialized to store 32-bit signed integers. Queues are simple and only have a few operations.

The data structure should support the following operations:

- Default construction - initializes an empty instance of a queue. This is not tested explicitly here!
- Assignment and copy construction - must properly copy the contents of another instance of the data structure. Ensure the two instances do not share memory afterwards!
- Push - must insert an element to the back of the queue.
- Pop - must remove the front element off of the queue.
- Front - must provide access to the front element of the queue.
- Size - must provide the size of the queue.

You must avoid any memory leaks or other memory errors in your implementation.

### Input

The input starts with a line containing an integer  $q$ , representing the number of lines that follow. Each line will represent an operation that is run. The lines have the following format: `<id> <operation> [arg]`

The instance ID will be an integer from 1 to 1,000, representing an instance of the data structure. Each instance should be default initialized at the start as an empty queue.

The operations are the following:

- `a` - construct a copy of queue, takes integer argument for the instance ID of the queue to copy
- `+` - push a value to the queue, takes integer argument for the value to push
- `-` - pop the queue, no additional arguments
- `f` - output the front element of the queue, see output section
- `s` - output size of queue, see output section

You may assume requested operations will not cause an error in a correctly implemented data structure. For example, a pop operation on an empty queue will not occur in the input.

### Output

For each front operation, output a line with the front element of the queue.

For each size operation, output a line with the size of the queue.

#### Sample Input 1

```
7
1 + 1
1 f
1 + 2
1 f
1 s
1 -
1 s
```

#### Sample Output 1

```
1
1
2
1
```

**Sample Input 2**

```
6
1 + -9
1 + 6
1 + 42
2 + -5
1 s
2 s
```

**Sample Output 2**

```
3
1
```