Doubly Linked List
Problem ID: doublylinkedlist

Write your own implementation of a doubly linked list, specialized to store 32-bit signed integers. Doubly linked lists are node based structures that allow traversing back and forth by following the links on the nodes.

Track a cursor (or current node) for each list which we will use the reference point for operations. The cursor should initially be the sentinel node of the empty instance.

The data structure should support the following operations:

- Default construction - initializes an empty instance with a reasonable capacity. 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!
- Front - return the first node of the list, not a sentinel node.
- Back - return the last node of the list, a sentinel node.
- Insert - must insert a node with the given element before the cursor, returning the newly inserted node.
- Erase - must erase the cursor node, returning the node after the erased node.
- Predecessor - return the node before the cursor node
- Successor - return the node after the cursor node
- Size - return the size of the instance

The cursor should **NOT** be a member of the instance, as in real world applications you may have more than one per list. The operations using the cursor nodes must take a node as an input parameter. After assignment, the cursor should be reset to the sentinel node of the instance. After insert, erase, predecesser, and successor operations, the cursor is set to the return value of the operation in question.

Note, in node based structures, some operations may be better suited as members of the node object, rather than the data structure object.

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 empty.

The operations are the following:

- `a` - construct a copy of the array, takes integer argument for the instance ID of the doubly linked list to copy
- `f` - front, no additional arguments
- `b` - back, no additional arguments
- `i` - insert, takes one integer argument, the element to insert
- `e` - erase, no additional arguments
- `>` - successor, no additional arguments
- `<` - predecessor, no additional arguments
- `g` - element access, read, see output section
- `s` - element access, write, takes integer argument, the element to write into the cursor node
- `z` - size, no additional arguments, see output section

You may assume requested operations will not cause an error in a correctly implemented data structure. For example, a get or erase operation on the last (sentinel) node will not occur in the input.

Output

For each get operation, output a line with the element of the instance's cursor node.

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

Sample Input 1

```
14
1 i 1
1 i 2
1 i 3
1 i 4
1 i 5
1 g
1 >
1 g
1 >
1 g
1 >
1 g
1 e
1 g
```

Sample Output 1

```
5
4
3
2
1
```

Sample Input 2

```
18
1 i 5
2 i 3
2 a 1
2 <
2 g
1 a 1
1 i 5
1 z
1 e
2 b
2 i 7
2 s 1337
2 f
2 i -42
2 g
2 >
2 >
2 g
```

Sample Output 2

```
5
2
-42
1337
```