Dynamically Sized Array
Problem ID: dynamicallysizedarray

Write your own implementation of a dynamically sized array, specialized to store 32-bit signed integers. Dynamically sized arrays have a size, representing the number of elements stored, and a capacity, representing the storage space available for elements.

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!
- Push back - must insert and element behind the currently last element.
- Pop back - must remove the last element.
- Insert - must insert an element at the given index, other elements must retain relative order.
- Erase - remove element at given index, other elements must retain relative order with no gaps between elements.
- Element access - must provide access to element at a given index for both read and write operations.
- Resize - sets the size of the array to the given value, if it grows then new values get default values. There is no requirement that this affects the capacity.
- Reserve - ensures there is at least the given amount of slots for values. This should never affect the elements in the data structure. This is not tested explicitly here!

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> [arg1] [arg2]`

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 with some reasonable initial capacity.

The operations are the following:

- `a` - construct a copy of the array, takes integer argument for the instance ID of the array to copy
- `+` - push back, takes integer argument for the value to push back
- `-` - pop back, no additional arguments
- `i` - insert, takes two integer arguments, the index where it should be placed and the element to insert
- `e` - erase, takes one integer argument, the index of the element to erase
- `g` - element access, read, takes integer argument for the requested index, see output section
- `s` - element access, write, takes two integer arguments, the requested index and the element to write into it
- `r` - resize, takes one integer argument, the desired resulting size
- `p` - print the instance, 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 array will not occur in the input.

Output

For each print operation, output the information for the data structure. First output a line with an integer s, the size of the instance. Then output a line with s space separated integers, the array elements in the order they appear in the array.

For each get operation, output a line with the element at the requested index.

Sample Input 1

```
7
1 + 1
1 + 2
1 + 3
1 + 4
1 + 5
1 p
2 p
```

Sample Output 1

```
5
1 2 3 4 5
0
```

Sample Input 2

```
10
1 r 8
1 p
1 s 6 42
1 p
1 -
1 p
1 -
1 p
1 r 7
1 p
```

Sample Output 2

```
8
0 0 0 0 0 0 0 0
8
0 0 0 0 0 0 42 0
7
0 0 0 0 0 0 42
6
0 0 0 0 0 0
7
0 0 0 0 0 0 0
```

Sample Input 3

```
19
1 + 33
1 g 0
1 r 5
1 p
2 i 0 -14
2 p
2 a 1
2 p
2 a 2
2 p
1 + 1
1 + 3
1 + 4
1 i 1 2
1 p
1 e 2
1 g 5
1 r 3
1 p
```

Sample Output 3

```
33
5
33 0 0 0 0
1
-14
5
33 0 0 0 0
5
33 0 0 0 0
9
33 2 0 0 0 0 1 3 4
1
3
33 2 0
```