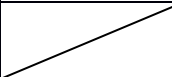
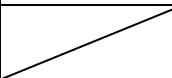
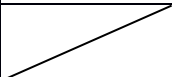
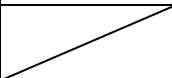


Reasoning Table for operationX

```
void operationX (ContainerC
  //! updates p
  //! clears y
  //! requires |p| > 1
  //! ensures  p = #p *
```

Append 0 to all variable names that appear in Assume's State 0.

Note: the state numbers allow us to talk about each of the values that a variable takes on throughout the lifetime of the operation.

S	Code	Assume	Confirm
0		$ p0 > 1$	$self \neq \langle \rangle$
	$p.op1(y);$	/ / / / / / / /	/ / / / / / / /
1		$self = \#self * \langle \#x \rangle$	$C1$
	...	/ / / / / / / /	/ / / / / / / /
k		Ak	Ck
	...	/ / / / / / / /	/ / / / / / / /
N		AN	$p = \#p * \langle \#y \rangle$

Reference:

```
void op1 (T& x);
  //! updates self
  //! clears x
  //! requires: self != <>
  //! ensures:  self = #self * <x>
```

Reasoning Table for operationX

```
void operationX (ContainerOfT& p, T& y);  
  /// updates p  
  /// clears y  
  /// requires |p| > 1  
  /// ensures p = #p * <#y>
```

In state N's Confirm column, append 0 to all # variables (i.e., incoming) and eliminate the #

S	Code	Assume	Confirm
0		p0 > 1	self /= <>
	p.op1(y);	/ / / / / / / /	/ / / / / / / /
1		self = #self * <#x>	C1
	...	/ / / / / / / /	/ / / / / / / /
k		Ak	Ck
	...	/ / / / / / / /	/ / / / / / / /
N		AN	p = p0 * <y0>

Reference:

```
void op1 (T& x);  
  /// updates self  
  /// clears x  
  /// requires: self /= <>  
  /// ensures: self = #self * <x>
```

Reasoning Table for operationX

```
void operationX (ContainerC5T& self, T& x) {
    //! updates p
    //! clears y
    //! requires |p| > 1
    //! ensures
```

In state N's Confirm column, append *N* to outgoing variables. For example, if *N* = 5, then append 5 to outgoing variable

```
p = #p * <#y>
```

S	Code	Assume	Confirm
0		p0 > 1	self /= <>
	p.op1(y);	/ / / / / / / /	/ / / / / / / /
1		self = #self * <#x>	C1
	...	/ / / / / / / /	/ / / / / / / /
k		Ak	Ck
	...	/ / / / / / / /	/ / / / / / / /
N		AN	pN = p0 * <y0>

Reference:

```
void op1 (T& x);
    //! updates self
    //! clears x
    //! requires: self /= <>
    //! ensures: self = #self * <x>
```

Reasoning Table for operationX

```
void operationX (ContainerOfT& p, T& y);  
  /// updates p  
  /// clears y  
  /// requires   |p| > 1  
  /// ensures   p = #p * <#y>
```

For the *requires* clause of a called operation:
Do variable substitution
Append the state number that is previous to the call. In this example, append 0

S	Code	Assume	Confirm
0		$ p_0 > 1$	$p_0 \neq \langle \rangle$
	<code>p.op1(y);</code>	////////	////////
1		<code>self = #self * <#x></code>	<i>C1</i>
	...	////////	////////
<i>k</i>		<i>Ak</i>	<i>Ck</i>
	...	////////	////////
<i>N</i>		<i>AN</i>	$p_N = p_0 * \langle y_0 \rangle$

Reference:

```
void op1 (T& x);  
  /// updates self  
  /// clears x  
  /// requires: self != <>  
  /// ensures:  self = #self * <#x>
```

Reasoning Table for operationX

```
void operationX (ContainerOfT& p, T& y);  
  /// updates p  
  /// clears x  
  /// requires   |p| > 1  
  /// ensures   p = #p * <#y>
```

For the *ensures* clause of a called operation:
Do variable substitution
Append the previous state number to incoming variables (0 in this example)
Append subsequent state number to outgoing variables (1 in this example)

S	Code	Assume	
0		$ p_0 > 1$	$p_0 \neq \langle \rangle$
	p.op1(y);	////////	////////
1		$p_1 = p_0 * \langle y_0 \rangle$	C1
	...	////////	////////
k		Ak	Ck
	...	////////	////////
N		AN	$p_N = p_0 * \langle y_0 \rangle$

Reference:

```
void op1 (T& x);  
  /// updates self  
  /// clears x  
  /// requires: self != <>  
  /// ensures: self = #self * <#x>
```