A Detailed Look at *append*'s Reasoning Table

append's Reasoning Table

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT - Code

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT - States

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 0 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 0 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append's RT – State 1 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append's RT – State 1 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 1 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td>true</td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 2 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append's RT – State 2 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 2 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			r2 = r1 g2 = g1
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append's RT – State 2 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 3 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 3 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td>true</td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append’s RT – State 4 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 4 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 4 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append's RT – State 5 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append's RT – State 5 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>
			$\boxed{\text{\|g5\| = 0}}$	

append's RT – State 5 Assume

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

append’s RT – State 5 Confirm

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>

append's RT – State 5 Confirm

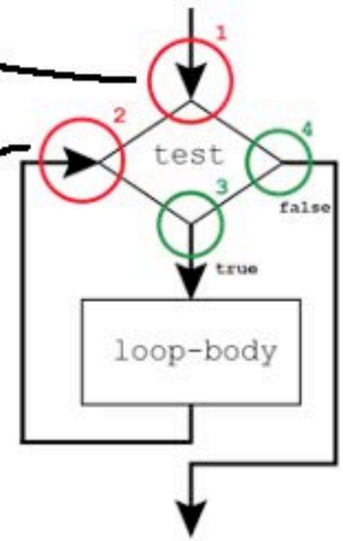
```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

S	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0
	while(g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases |g|			
1		|g1| > 0 ^ r1 * g1 = r0 * g0		
	T y;			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	g.dequeue(y);			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	r.enqueue(y);			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0
	}			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0		r5 = r0 * g0 ^ g5 = <>

Loop Invariant Requires Confirming

In locations **1** and **2** is where we must prove the loop invariant holds

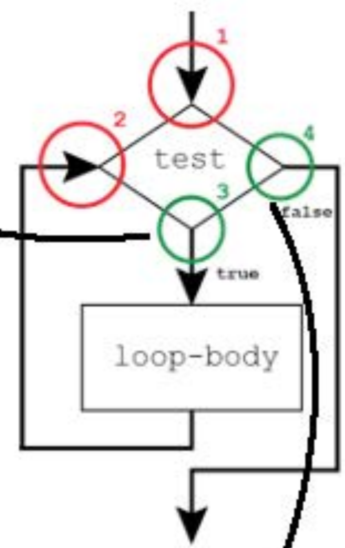
<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>7</td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>



Loop Invariant Assumed True

In locations **3** and **4** is where we get to assume the loop invariant holds

<table>
<tr><th>S</th><th>Code</th><th colspan="2">Assume</th><th>Confirm</th></tr>
<tr><td>0</td><td></td><td colspan="2">true</td><td>r0 * g0 = r0 * g0</td></tr>
<tr><td></td><td>while(g.length() > 0) {
//! updates g, r
//! maintains
//! r * g = #r * #g
//! decreases |g|</td><td colspan="2"></td><td></td></tr>
<tr><td>1</td><td></td><td colspan="2">|g1| > 0 ^
r1 * g1 = r0 * g0</td><td></td></tr>
<tr><td></td><td>T y;</td><td colspan="2"></td><td></td></tr>
<tr><td>2</td><td></td><td>T.Init(y2)</td><td>Unchanged
r, g</td><td>g2 /= <></td></tr>
<tr><td></td><td>g.dequeue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>3</td><td></td><td>g3 = g2[1,|g2|) ^
<y3> = prefix of g2</td><td>Unchanged
r</td><td></td></tr>
<tr><td></td><td>r.enqueue(y);</td><td colspan="2"></td><td></td></tr>
<tr><td>4</td><td></td><td>T.Init(y4) ^
r4 = r3 * <y3></td><td>Unchanged
g</td><td>|g4| < |g1| ^
r4 * g4 = r0 * g0</td></tr>
<tr><td></td><td>}</td><td colspan="2"></td><td></td></tr>
<tr><td>5</td><td></td><td colspan="2">~(|g5| > 0) ^
r5 * g5 = r0 * g0</td><td>r5 = r0 * g0 ^
g5 = <></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>7</td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
</table>



append's RT – Major Assertions

```
void appendV1 (QueueOfT& r, QueueOfT& g) // Using r for receiver, g for giver
//! updates r
//! clears g
//! requires: true
//! ensures: r = #r * #g
```

s	Code	Assume		Confirm
0		true		r0 * g0 = r0 * g0 ①

	`while(g.length() > 0) {`
	`//! updates g, r`
	`//! maintains`
	`//! r * g = #r * #g`
	`//! decreases |g|`			

1		|g1| > 0 ^ r1 * g1 = r0 * g0 ③		
	`T y;`			
2		T.Init(y2)	Unchanged r, g	g2 /= <>
	`g.dequeue(y);`			
3		g3 = g2[1,|g2|) ^ <y3> = prefix of g2	Unchanged r	
	`r.enqueue(y);`			
4		T.Init(y4) ^ r4 = r3 * <y3>	Unchanged g	|g4| < |g1| ^ r4 * g4 = r0 * g0 ②
	`}`			
5		~(|g5| > 0) ^ r5 * g5 = r0 * g0 ④		r5 = r0 * g0 ^ g5 = <>
7				

Up Next – *append*'s VCs

1. Using the assertions in this RT to generate *append*'s VCs
2. Proving the VCs to prove *append* is correct, i.e., meets its spec