

Loop Invariants: Part 2

These slides courtesy of
Paolo Bucci, Bruce Weide, et al.
The Ohio State University's
Department of Computer Science
With some minor modifications by Dr. Holly



Example

```
void append(QueueOfT& r, QueueOfT& g)  
//! Concatenates (“appends”) g to the end of r  
//! updates: r  
//! clears: g  
//! ensures:  $r = \#r * \#g$ 
```

append's Function Body

```
while (g.length() > 0) {  
    //! updates g, r  
    //! maintains  $r * g = \#r * \#g$   
    //! decreases  $|g|$   
    T y;  
    g.dequeue(y);  
    r.enqueue(y);  
} // end while
```

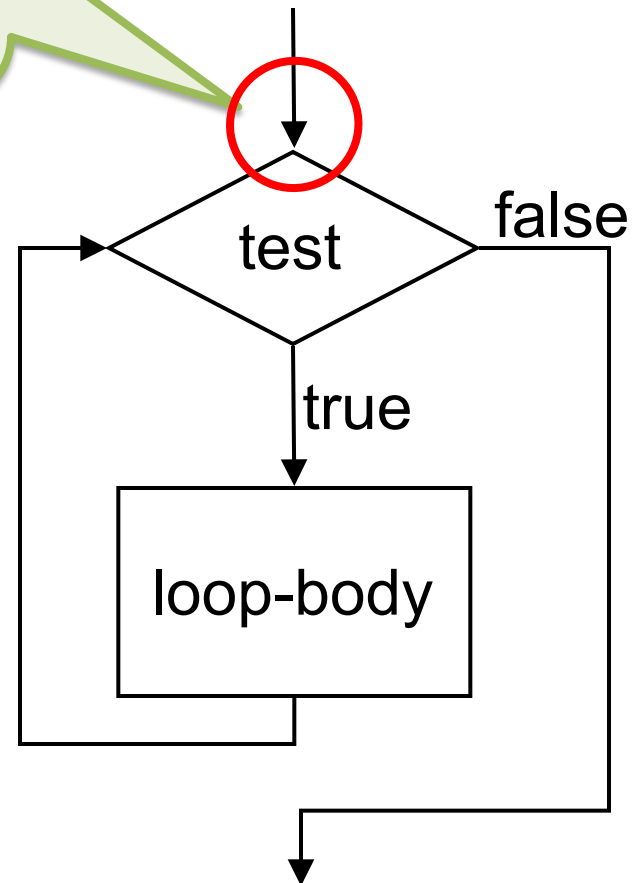
Control Flow

append's loop invariant

$r * g = \#r * \#g$

holds here

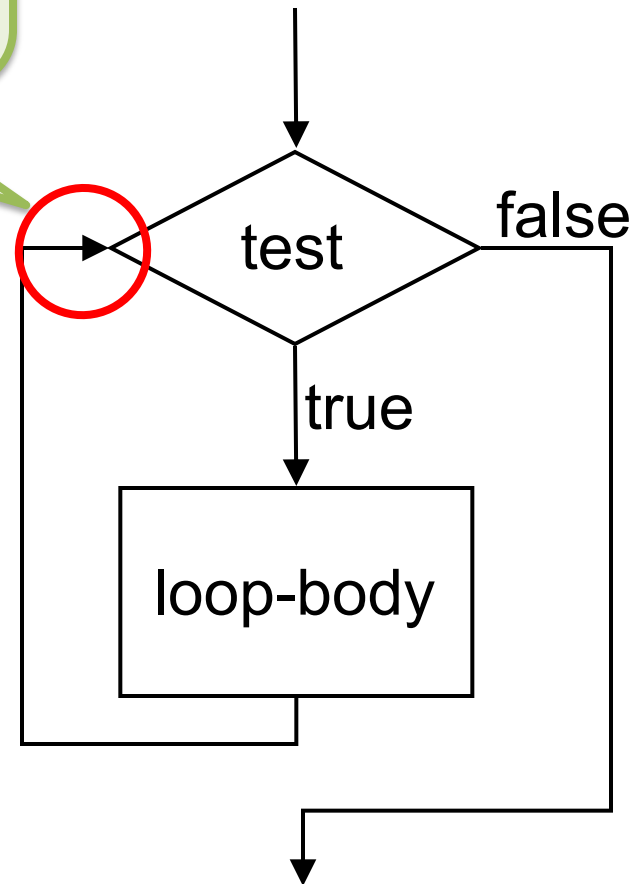
```
while (test) {  
    loop-body  
}
```



Control Flow

... and here

```
while (test) {  
    loop-body  
}
```



Using a Loop Invariant

If you have a strong enough loop invariant, you can do two things with it:

1. You can ***trace over*** a loop in a single step, and predict the values of the variables when it terminates—you do this without tracing through the loop body

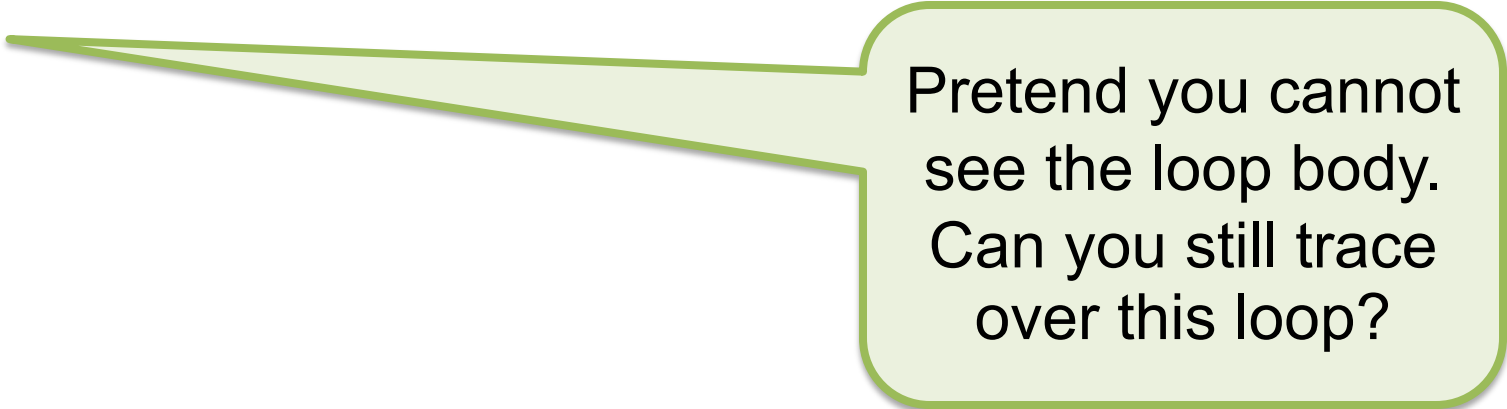
Using a Loop Invariant

If you have a strong enough loop invariant, you can do two things with it:

2. Use the loop invariant in the Assume column of a *reasoning table* in the state just after the loop when doing a proof of an operation containing a loop

Example of a Trace Over

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



Pretend you cannot see the loop body.
Can you still trace over this loop?

	$r = \langle 1, 2, 3 \rangle$ $g = \langle 4, 5, 6 \rangle$
<pre>while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases g </pre>	
...	
}	
	$r =$ $g =$

When execution reaches
this point, we know two
facts...

	$r = \langle 1, 2, 3 \rangle$ $g = \langle 4, 5, 6 \rangle$
<pre>while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases g ... }</pre>	<p>We know (1) the loop invariant is true, so:</p> $r * g = \#r * \#g$
	$r =$ $g =$

	$r = \langle 1, 2, 3 \rangle$ $g = \langle 4, 5, 6 \rangle$
<pre>while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases g </pre>	
...	
}	
	$r =$ $g =$

We also know (2) the loop condition is false, so:

$$|g| \leq 0$$

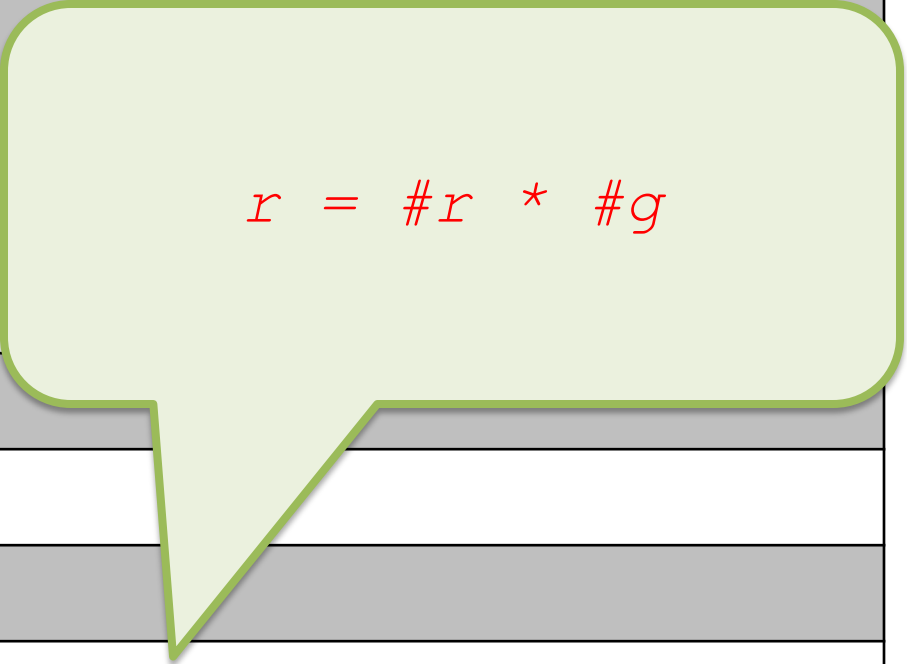
	$r = \langle 1, 2, 3 \rangle$ $g = \langle 4, 5, 6 \rangle$
<pre> while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases g </pre>	
...	
}	
	$r = \langle 1, 2, 3, 4, 5, 6 \rangle$ $g = \langle \rangle$

Based on these 2 facts, the only values the variables can possibly have at this point are these

	$r = \langle 1, 2, 3 \rangle$ $g = \langle 4, 5, 6 \rangle$
<pre> while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases g </pre>	
...	
}	
	$r = \langle 1, 2, 3, 4, 5, 6 \rangle$ $g = \langle \rangle$

$r * g = \#r * \#g$ and
 $|g| \leq 0$

	$r = \langle 1, 2, 3 \rangle$ $g = \langle 4, 5, 6 \rangle$
<pre>while (g.length() > 0) { //! updates g, r //! maintains //! r * g = #r * #g //! decreases g </pre>	
...	
}	
	$r = \langle 1, 2, 3, 4, 5, 6 \rangle$ $g = \langle \rangle$



$$r = \#r * \#g$$

Justification for Fact 1

- The loop invariant is true just after the loop terminates—*if* the code that evaluates the loop condition does not change the value of any variable appearing in the loop invariant

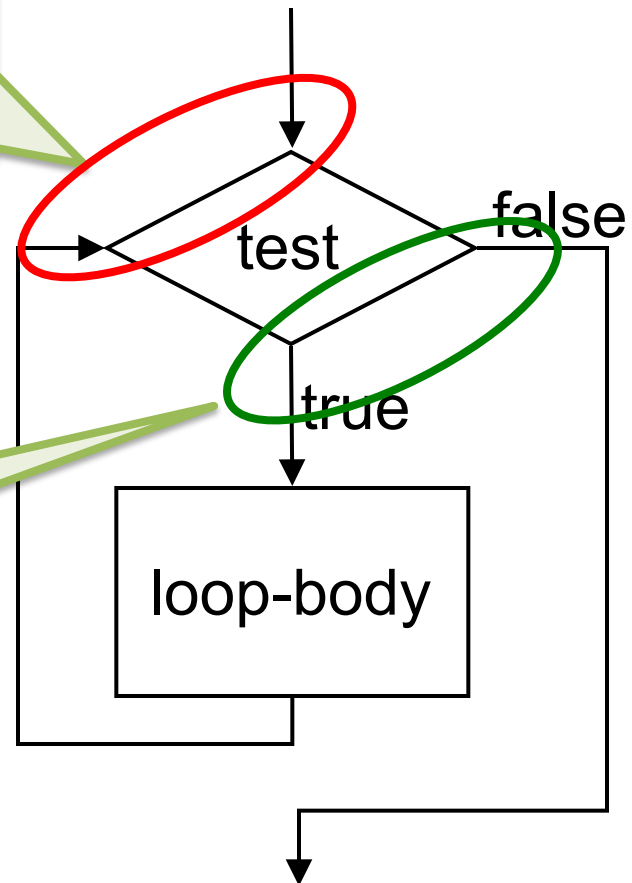
The Loop Invariant Picture

If the loop invariant is true
at the two red points,
and “test” *updates* nothing...

```
↑ while (test) { ↑  
    loop-body
```

```
↑ } ↑
```

...then the loop invariant is true
at the two green points.



Engineering Rule Based on Fact 1

- **Engineering Rule:** Code that evaluates the loop condition should never update any variables appearing in the loop invariant
 - An easy way to achieve this:
the test should not update *any variables at all*

Justification for Fact 2

- The loop does not terminate until and unless the loop condition is false

Justification for Fact 2

- The loop does not terminate until and unless the loop condition is false
 - However, a loop *might never terminate*; so you need to show that it actually does terminate

Loop Termination

- To show that a loop terminates, it is sufficient to provide a ***progress metric***

Loop Termination

- To show that a loop terminates, it is sufficient to provide a *progress metric*
- Which has these 3 properties:
 1. It is an integer-valued function of the variables involved in the loop's body
 2. It is always non-negative
 3. It always decreases when the loop body is executed once

Example: Function Body

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

The *decreases* keyword states that this loop's body causes the length of `g` to decrease each execution of the loop body

append's Function Body

```
while (g.length() > 0) {  
    //! updates g, r  
    //! maintains  $r * g = \#r * \#g$   
    //! decreases  $|g|$   
    T y;  
    g.dequeue(y);  
    r.enqueue(y);  
} // end while
```

Conclusion

- Even if you do not choose to *write down* a loop invariant or progress metric, at least *think about* loops in these terms

Conclusion

- Even if you do not choose to *write down* a loop invariant or progress metric, at least *think about* loops in these terms
- This approach will help you avoid errors and bad practices in loop code, such as:
 - Off-by-one errors
 - Wrong/missing code in the loop body
 - Declarations of variables outside the loop that are only used inside the loop body