**Announcements**

- HW1 due Tuesday
  - You must clone a new repository, hw01
  - To submit answers, push to Git, then submit in Submitty
- If you have questions, please email us at csci2600@cs.lists.rpi.edu
- Use LMS discussion board

---

**Outline**

- Quiz 1
- Reasoning about loops (conclusion)
- Dafny lab (optional, 2pts extra towards HW)
- Specifications (next time)

---

**Quiz 1, String Comparison**

```java
String a = new String("RPI");
String b = new String("RPI");
System.out.println(a == b); // Yields false
```

```java
String a = "RPI";
String b = "RPI";
System.out.println(a == b); // Yields true!
```

---

**So Far**

- We discussed reasoning about code
  - Forward reasoning and backward reasoning
- Hoare Logic
  - Hoare Triples
  - Rule for backward reasoning
    - Assignment
    - Sequence
    - If-then-else
    - Method call
- Reasoning about loops

---

**Reasoning about Loops, Conclusion**
Reasoning About Loops

total correctness = partial correctness + termination

1. Partial correctness
   • "Guess", then prove loop invariant
   • Loop invariant and loop exit condition must imply the postcondition
   • This gives us: "If the loop terminated then the postcondition did hold". But does the loop terminate?

2. Termination
   • "Guess" decrementing function D.
     (1) D >= 0, (2) strictly decreases, (3) D at 0 along with the loop invariant must imply loop exit condition.

A "tedious" Invariant

Inv: \( \text{sum} = a[0]+a[1]+\ldots+a[i-1] \)

Precondition: \( \text{len} \geq 1 \) \&\& \( \text{a.length} = \text{len} \)
int sum = a[0];
int i = 1;
while (i < len)
   invariant sum = a[0]+\ldots+a[i-1] \&\& i<len
   { sum = sum + a[i];
     i = i+1;
   }
Postcondition: sum = a[0]+\ldots+a[a.length-1]

Another Factorial

Precondition: \( t \geq 0 \)
int r = 1;
t = t;
while (n != 0) {
   r = r*n;
   n = n-1;
}
Postcondition: r = t!

Let's Catch the Bug

Inv: \( \text{sum} = a[0]+a[1]+\ldots+a[i] \)

Precondition: \( \text{len} \geq 1 \) \&\& \( \text{a.length} = \text{len} \)
int sum = a[0];
int i = 1;
while (i <= len)
   invariant sum = a[0]+\ldots+a[i-1] \&\& i<=len+1
   { sum = sum + a[i];
     i = i+1;
   }
Postcondition: sum = a[0]+\ldots+a[a.length-1]

"Interesting" Invariant

As the induction variable \( i \) moves through the array, \( \text{sum} \) contains the correct partial result.
At the end of the array, \( \text{sum} \) contains the correct total
Similar invariants for min, max, avg, sorting algorithms, etc.
These invariants are easy to guess.
Interesting Invariant

Integer Division
Precondition: \( x \geq 0 \) \&\& \( y > 0 \)
\[
\begin{align*}
    r & = x; \\
    q & = 0; \\
    \text{while } (y \leq r) \{ \\
    & \quad r = r-y; \\
    & \quad q = q+1; \\
    \}
\end{align*}
\]
Postcondition: \( x = y \cdot q + r \) \&\& \( r < y \)

Dafny Lab

Trailing Zeros
Precondition: \( x > 0 \)
\[
\begin{align*}
    \text{zeros} & = 0; \\
    y & = x; \\
    \text{while } (y \% 10 == 0) \{ \\
    & \quad y = y/10; \\
    & \quad \text{zeros} = \text{zeros} + 1; \\
    \}
\end{align*}
\]
Postcondition: \( x = y \cdot 10^\text{zeros} \) \&\& \( y \% 10 != 0 \)

Dafny Lab

You may have to add a Dafny function to state the postcondition. E.g., pow2(n):
\[
\text{function pow2(n: nat): nat} \{ \text{if } n == 0 \text{ then 1 else } 2 \cdot \text{pow2}(n-1) \}
\]

Note: it may be easier to verify the program if you specify input and output parameters as \text{nat}, not int.

method trailingZeros(x: nat) returns (y: nat, zeros: nat)
requires ...
ensures ...
\[
\begin{align*}
    \{ \\
    \quad \ldots \\
    \}
\end{align*}
\]