Announcements

- HW1 due Friday
  - To submit answers, first commit to SVN, then submit through Homework 1 in Homework Server

- If you have questions, please email csci2600@cs.lists.rpi.edu

Outline

- Quiz 1
- Reasoning about loops (conclusion)
- Dafny lab (optional, 2pts extra towards HW2)
- Specifications

Quiz 1, String Comparison

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

String a = "car";
String b = "car";
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 assignment
  - Rule for sequence
  - Rule for if-then-else
  - Rule for method call
  - Reasoning about loops
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 holds". But does the loop terminate?

2. Termination
   - (Informally) "Guess" decrementing function D. Each iteration decrements D, D at 0 along with the loop invariant must imply loop exit condition

A "tedious" Invariant

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

Precondition: \( \text{len} \geq 0 \) \&\& \( a.\text{length} = \text{len} \)

int \( \text{sum} = 0; \)
int \( i = 0; \)
while (\( i < \text{len} \))
   in\( \text{variant} \) \( \text{sum} = a[0]+a[i-1] \) \&\& \( i < \text{len} \)
   \{ \( \text{sum} = \text{sum} + a[i]; \)
   \( i = i+1; \)
\}
Postcondition: \( \text{sum} = a[0]+a[\text{length}-1] \)

Let's Catch the Bug

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

A "tedious" Invariant

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

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

Another Factorial

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

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

Dafny Lab

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

Outline

- Specifications
  - Benefits of specifications
  - Specification conventions
    - Javadoc
    - PoS specifications
    - JML
  - Specification style
  - Specification strength

Specifications

- A specification consists of a preconditon and a postcondition
  - Precondition: conditions that hold before method executes
  - Postcondition: conditions that hold after method finished execution (if precondition held)

Specifications

- A specification is a contract between a method and its caller
  - Obligations of the method (implementation of the specification): agrees to provide postcondition if preconditon held
  - Obligations of the caller (user of specification): agrees to meet the precondition and not expect more than the promised postcondition

- A specification is an abstraction
Example Specification

Precondition: \( \textit{len} \geq 0 \&\& \textit{a.length} = \textit{len} \)
Postcondition: \( \textit{result} = \textit{a[0]}+\ldots+\textit{a[a.length-1]} \)

```java
int sum(int[] a, int len) {
    int sum = 0;
    int i = 0;
    while (i < len) {
        sum = sum + a[i];
        i = i+1;
    }
    return sum;
}
```

For our purposes, we will be writing specifications that are a bit less formal than this example. Mathematical rigor is welcome, but not always necessary.

Benefits of Specifications

- Precisely documents method behavior
- Promotes modularity
- Client relies on description in specification, no need to know the implementation
- Method must support specification, but its implementation is free otherwise!

So, Why Not Just Read Code?

Code is complicated
- Gives a lot more detail than client needs
- Understanding code is a burden

Code is ambiguous
- What is essential and what is incidental (e.g., result of an optimization)

What About Comments?

```java
// method checks if \textit{part} appears as a subsequence in \textit{src}
boolean sub(List\langle T\rangle src, List\langle T\rangle part) {
    int part_index = 0;
    for (Object o : src) {
        if (o.equals(part.get(part_index))) {
            part_index++;
            if (part_index == part.size()) {
                return true;
            }
        } else {
            part_index = 0;
        }
    }
    return false;
}
```

Comments are important, but insufficient. They are informal and often ambiguous.
Specifications Are Concise and Precise

- Unfortunately, lots of code lacks specification
- Programmers guess what code does by reading the code or running it
- This results in bugs and/or complex code with unclear behavior
- Specification inference is an active area of research

So, What’s in sub’s Specification?

Choice 1:
// sub returns true if part is a subsequence of src; it returns false otherwise.
Choice 2:
// src must be non-null
// If src is the empty list, then sub returns false
// Requirements on part too...
// If there is a partial match in the beginning, sub returns false, even if there is a full match later.
E.g. sub([1, 2, 1, 2, 1, 3], [1, 2, 1, 3]) is false.

What’s in sub’s Specification?

- This complex specification is a red flag
- Rule: it is better to simplify design and code rather than try to describe complex behavior!
- If you end up writing a complicated specification, redesign and rewrite your code
--- either something is wrong or code is extremely complex and hard to reason about
(Mike Ernst calls this “a sneaky fringe benefit of specifications”)

Goal of Principles of Software

- One goal of Principles of Software is to instill a discipline of writing concise and precise specifications

Outline

- Specifications
- Benefits of specifications
- Specification conventions
  - Javadoc
  - PoS specifications
  - JML
- Specification style
- Specification strength