Slide 1

## Administrivia

- Homework 6 on the Web (finally). Due in a week.
- We'll stay with six quizzes and have the next one next Tuesday, over material in chapter 3. Homework 6 should give some clues about kinds of problems.
- (Review minute essay from 4/16.)

Slide 2

## Program Correctness — Recap

- One way to increase your confidence that a program "works" is to test it. That only goes so far, however. Another approach is to reason about it somehow. This can be done very formally (at least for small programs), or the ideas can be applied informally.
- The formal approach involves first defining what we mean by "program works" (precondition/postcondition, Hoare triples) and then gives rules for assignment, sequential composition, if/then/else, and loops.

Slide 3

## Program Correctness and Loops — Review

- We'll write loops in this form

  **while** $B$ **do**
  $P$
  **end while**

  After the loop terminates (assuming it does), what do we know about $B$? True or false?

- We also need the notion of a "loop invariant" — a predicate that, if true before we execute the loop body is true again after. More formally, $Q$ is an invariant for the above loop if

$$\{\ Q\ \wedge\ B\ \}\ \ P\ \ \{\ Q\ \}$$

- Now we can state the rule for loops . . .

Slide 4

## Program Correctness and Loops

- For program $P_1$ as follows

  **while** $B$ **do**
  $P$
  **end while**

  and $Q$ an invariant of the loop in $P_1$, we can say that

$$\{\ Q\ \}\ \ P_1\ \ \{\ Q\ \wedge\ B'\ \}$$

- We could prove this using induction (on the number of trips through the loop).
- The idea is to choose $Q$ such that the postcondition in the above triple ($Q\ \wedge\ B'$) is useful — i.e., helps establish something we want to be true after the loop.

Slide 5

## Trivial Example

- Suppose we have

  **while** $x > 0$ **do**

  $x := x - 1$

  **end while**

  with $x$ an integer variable.

- Show that if before the loop $x$ is a non-negative integer, after the loop $x = 0$.

Slide 6

## Correctness of Loops, Continued

- The textbook isn't very explicit about this, but strictly speaking we have something else to prove — that the loop terminates!
- Can do this with a "metric" (think "measure") — integer function of program variables that decreases every time through the loop, and when it's less than or equal to zero the loop stops.
- In the silly example, we could use what? (The value of X.)

Slide 7

## Things to Notice about Loop Invariants

- They're not unique — could come up with many "invariants" for a given loop. (This is true about preconditions in general.)
- The goal is to find one that's "useful" — if true at end of the loop with loop test false, helps us prove desired postcondition.
- Sometimes helps to think in terms of "what do the variables mean?"
- Writing down a loop invariant can help (e.g., to avoid off-by-one errors) even if you don't do a complete formal proof.

Slide 8

## Silly Example

- Example — silly program to compute $z = x \times y$ by repeated addition (for $x$ and $y$ positive integers):

  $i := 0; z := 0;$
  **while** $i < x$ **do**
  $z := z + y; i := i + 1$
  **end while**

Slide 9

## Example — GCD

- Another example — Euclid's algorithm for finding GCD (greatest common divisor, a.k.a. largest common factor) of $a$ and $b$ (where $a$ and $b$ are positive integers):

  $i := a; j := b;$
  **while** $j \neq 0$ **do**
  $q := i/j; r := i\%j;$
  $i := j; j := r;$
  **end while**

  At end, $i = gcd(a, b)$. It does?! Yes, and we can prove it, even if we don't quite understand *why*. Next slide . . .

Slide 10

## GCD, Continued

- Proposed invariant (using textbook's subscripting notation — possibly clearer when we need to talk about old/new values of variables):

$$gcd(i_n, j_n) = gcd(a, b)$$

- Prove that it's an invariant using the following lemma:
  If $a = qb + r$, then

$$gcd(a, b) = gcd(b, r)$$

Slide 11

## GCD, Continued

- Strictly speaking we also need to show that the loop stops. The simplest way to do this is to require that initially $a > b$ and then observe that $j$ is a non-negative integer (why?) and gets smaller (why?) on every trip through the loop, and the algorithm stops when it becomes zero.
- (Is it reasonable to require that initially $a >$b? I say yes, since $gcd$ is commutative, and $gcd(a, a) = a$.

Slide 12

## Proofs of Program Correctness — Recap/Evangelism

- Many examples we looked at are trivial — mostly because they're all we can do in the time we have. (Textbook's proof that Euclid's algorithm works is a notable exception.) Keep in mind, though:
  - How to make this practical, and/or how to have it done by a smart program, are research topics.
  - In my opinion/experience, applying these ideas informally helps you "reason about programs". ("What do you know about the program variables at this point?" "What is this variable supposed to represent, and does the code support that?")
  - Similar ideas are very useful in reasoning about concurrent algorithms, which otherwise can be *very* tricky!

**Slide 13**

## Minute Essay

- None — sign in.