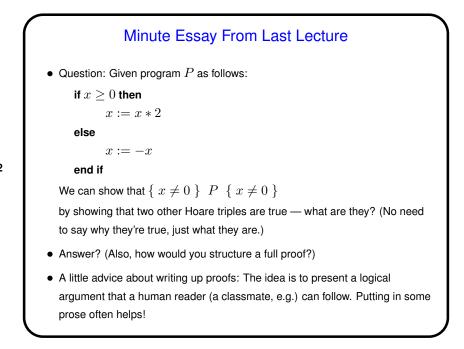


Slide 1



Slide 2

Program Correctness and Loops, Review • Our rule is this: For program P of the form while B do P_1 end while if we also have a "loop invariant" Q, such that $\{Q \land B \} P_1 \{Q\}$ then we can derive $\{Q\} P \{Q \land B'\}$ • Strictly speaking, we also have to prove that the loop does terminate — can do this by finding an integer function ("metric") that decreases every time through and when not positive means B is false.

Program Correctness and Loops, Continued
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.

Slide 4

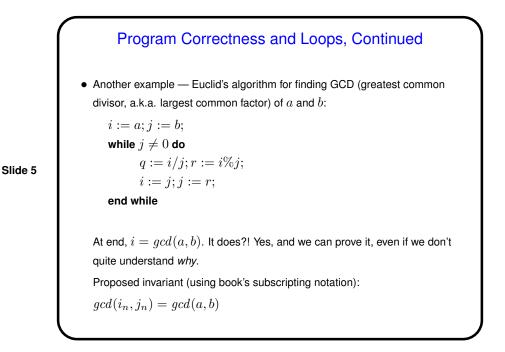
Sometimes helps to think in terms of "what do the variables mean?"
Writing down a loop invariant can help (e.g., to avoid off-by-one errors) even if you don't do a complete formal proof.

Example — silly program to compute z = x × y by repeated addition:

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

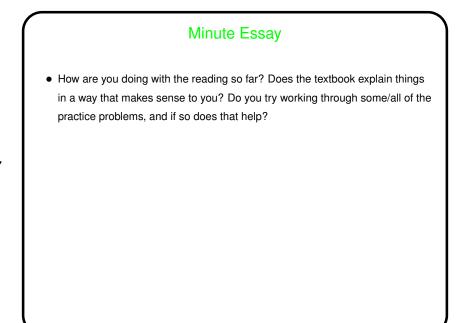
Slide 3

Slide 6



Proofs of Program Correctness, Recap
Many examples we looked at are trivial — mostly because they're all we can do in the time we have. Keep in mind, though:

How to make this practical, and/or how to have it done by a smart program, are subjects of ongoing research.
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 7