[1] The first topic this lecture is refinement, and refinement is the main idea in the course. Some people call this whole subject programming by refinement. The next best textbook for this course, after my own, of course, is named refinement calculus. So what is refinement? Specification P is refined by specification S means that whenever S is satisfied, so is P. Formally, [2] that's for all pre- and poststates, P is implied by S. If someone gives you a specification P and says please implement this, you can refine P, which means choosing an equal or stronger specification S, and implement S instead. That's because computer behavior that satisfies S also satisfies P, so the customer will be happy. For example, suppose the variables are x and y, of type integer. And the specification is [3] x prime is greater than x. The customer wants x increased. Yes, I know it's a stupid little example, but it gives the idea. One way to refine that specification is by x prime equals x plus 1 and y prime equals y. You had to decide by how much to increase x, and you decided to increase it by 1. And you had to decide what to do with y, and you decided to leave it alone. Refinement means making such decisions, reducing the nondeterminacy. The new specification is [4] equal to x gets x plus 1, and that is also refinement, because equality implies implication. The [5] next example is refining x prime less than or equal to x by if x equals 0 then leave x alone, else decrease x, which we could [6] write as a disjunction if we want. And [7] one more example. Make x bigger than y in the end, and make them both bigger than x was to start. That can be refined by y gets x plus 1 followed by x gets y plus 1. For the proof of this refinement, you can [8] rewrite the last assignment, and then use the substitution law to get [9] x prime equals x plus 2 and y prime equals x plus 1.

A [10] condition is a specification, or binary expression, that talks about just one state. If it's the [11] initial state or prestate, it's called an initial condition, or precondition. If it's the [12] final state or poststate, it's called a final condition, or postcondition. The [13] exact precondition for specification P to be refined by specification S is this formula, which is exactly the same as the refinement formula, except refinement says, for all prestates sigma and poststates sigma prime, P is implied by S, and this formula just quantifies over poststates sigma prime. Now P and S can talk about both sigma and sigma prime. This formula makes sigma prime local, and sigma is nonlocal, so this formula just talks about sigma, the prestate. And [14] there's a similar formula for the exact postcondition, where sigma is local and sigma prime is nonlocal. [15] A sufficient precondition is any precondition that implies the exact precondition. A necessary precondition is any precondition that is implied by the exact precondition. So the exact precondition is the necessary and sufficient precondition. And similarly for postconditions. Let's [16] look at an example with one integer variable x. We want to find the exact precondition for x prime greater than 5 to be refined by x gets x plus 1. First of all, x prime greater than 5 **is not** refined by x gets x plus 1. Adding 1 to x doesn't necessarily make it bigger than 5. The formula says [17] for all x prime x prime is greater than 5 if x gets x plus 1. Since there's only 1 variable, the assignment [18] is just x prime equals x plus 1. Now, do you remember we used a [19] one point law for an existential quantification? Well, there's a similar law for universal quantification, except that for *exists* it's a conjunction, and for *for all* it's an implication. So that's perfect for us, and we get [20] x plus 1 greater than 5, which is [21] x greater than 4. The exact precondition tells us, under what initial condition, does increasing x by 1 make it bigger than 5. And the answer is, of course, if x starts out bigger than 4. This is just a tiny example, but the same calculation works for large examples too. If you want to know under what condition some program works, you don't have to guess. You calculate. And you find out exactly when it works and when it doesn't. [22] x prime greater than 5 is *not* refined by x gets x plus 1, so there's 2 things you can do about it. One is to change the right side, and find something it is refined by. The other is to weaken the left side by [23] adding the exact precondition as antecedent. Or any sufficient precondition will do. Promise

a little less. Don't promise to make x bigger than 5. Promise that if x starts out bigger than 4, then it will become bigger than 5.

[24] Now we have a postcondition example. Find the exact postcondition for x greater than 4 to be refined by x gets x plus 1. The specification x greater than 4 isn't even implementable because there's no way a computer can make its input be greater than 4. But let's do it anyway and see what we get. [25] Here's what the formula says. And again, in one variable, the assignment [26] is just an equation. To use one point, we would need x equals something in the antecedent. So [27] that's easily arranged. And now [28] one point says it's x prime minus 1 greater than 4, which is [29] x prime greater than 5. So that tells us that, although adding 1 to x doesn't make the input be bigger than 4, if the result turns out to be bigger than 5, then we know the input was bigger than 4. And although [30] this isn't a refinement, if we just [31] weaken the problem with the antecedent we just calculated, we get a refinement. On the left, we can read that as saying if the output is greater than 5 then the input was greater than 4. Or we can use [32] the contrapositive law to turn the implication around. a implies b is the same as not b implies not a. So we can [33] rewrite the specification to say – if x starts out less than or equal to 4, then it will end up less than or equal to 5.

[34] There are some laws about conditions. In these laws, C is a precondition, and C prime is a postcondition, and if they occur in the same law, then they're the same except that C doesn't have primes on the variables, and C prime does. P and Q are any specifications, not necessarily conditions. [35] The first one says that a precondition that's conjoined to a dependent composition is really conjoined to the first part of the composition, because it's a precondition and it just talks about the initial state. [36] And the same thing if it's an antecedent. [37] And the next 2 laws say exactly the same thing about postconditions. When they're conjoined to or antecedent to a dependent composition, it's really to the last part of the composition, because they just talk about the poststate. [38] In the next one, we have a precondition conjoined to the second part of a dependent composition. Well the start of the second part is really the same as the end of the first part. [39] And in the last one, on the left we have a dependent composition. Behave according to P, and then according to Q. The law says, if we want, we can do more than P to start with. We can do P and C prime. And then we have an easier time after because we can assume C when doing Q. — Well, I don't remember these laws. I just remember that there are laws like these, and I look them up when I need them, which isn't very often. [40] The next 2 laws say what you already know. You can use a sufficient precondition or sufficient postcondition to weaken a specification and turn a non refinement into a refinement.

That's it for this lecture. Next lecture we get to do some programming.