Boolean Expressions and Functions

9-14-2011
Do you have any questions about the quiz?

Minute essay comments
- Roller skating class does allow in-line.
- The joy of programming and the agony of silly errors.
- Tabbing in and making tabs small.
- Putting things into your web space.
- Bias and mantissa.
- Are there limits for conditionals/Booleans?

Let's finish what we were doing last class.
I want to have a function that tells me if two squares intersect.

The function will be given the x and y location of the center of each square as well as the length of the side of each square.

It should return a Boolean telling if they intersect.
We talked about comparisons of values in the last class.

We can also combine Boolean expressions together using Boolean logic.

There are four Boolean operators:

- && for and
- || for inclusive or
- ^ for exclusive or
- ! for not
Short Circuit Operators

- The && and || operators are short circuit operators.
- This means that if the value is known after evaluating the first operand, the second operand won't be evaluated.
- This can prevent errors.
- Let's look at an example of this with division by zero.
Nesting if statements

- What you put in an if can be any expression or statement.
- As a result, you can put an if inside of another if.
- As we will see, Scala doesn't care what you nest inside of things. You write the logic that makes sense to you and says what you want to say.
Let's review the concept of functions from math.

In algebra a function would take one or more values and give you back a value. The values were generally numbers.

In higher level math this is generalized with things like sets.

In math functions the same input always leads to the same result.
The concept of a function is critically important to programming.

Functions can take one or more arguments and give us back values. (Most languages allow only one return value.)

Let's think of some examples of functions that we could write.
We declare functions in Scala using def. Here is the general form.

- def name(arg1:Type1, arg2:Type2, ...):Type = expression

The argument list can have zero or more elements. If there are zero even the parentheses can be left off.

- Function arguments must have types.
- The return type is optional, but it is recommended.
Why Functions?

- Functions are used in programs for a number of reasons.
  - Reduce code duplication. You can call the same function multiple times and only write it once.
  - Improve readability and maintainability. Good function names make it easier to read. Small functions are easier to test and debug.
  - Break problems down/problem decomposition.
Problem Decomposition

- Never solve a hard problem. If a problem is hard, break it into smaller problems that are easier. Repeat until you are only solving trivial problems.

- Top-down
  - This is the “normal” approach where you start with the full problem and break it into pieces.

- Bottom-up
  - Sometimes you realize that different trivial pieces will be useful and build up from those.
What are your thoughts so far on the book? Have you been reading? How much is it helping?

Remember, there is no class on Friday for the campus wide curriculum discussion.