I've run into two complications doing exponentiation with integers.

1) Since the built in pow(double, double) function uses other functions that return doubles, there's a chance that it can function improperly when you call it with two ints and expect an int. The result can be truncated, so you end up one short. Need we write our own integer power function?

2) In general, for all the operations, if we're given two ints we're to return an int value. However, this method doesn't really make sense with exponentiation, since exponentiation isn't closed on the set of integers (and doesn't have a neat built in method like division). Negative exponents yield fractions (e.g. $2^{-1} = 0.5$). Should we just truncate and return an int, or should we return a double in this case?

For the first issue, if both integers are positive then you can just call pow() with two integers, and then just add .5 to the result before you truncate. But this doesn't yield the correct result if either the base or exponent are negative.

I might have missed the boat on requiring the exponent of two ints to be type int. Is there somewhere Dr. Lusth said or in class that we cannot store the result of int $^\text{int}$ as a double? I probably just missed it somewhere!

I don't remember him saying explicitly, which is why I wasn't sure. As I said, for the rest of the operations if you have two ints you return an int, but that doesn't really make sense for this one. I'm hoping he responds and just lets us return a double; that'd make things really easy (and I may be able to turn this in on time for extra credit lol).
I'll allow either. Exponentiation between two ints can return either a reasonable int or a real.

Suppose it returns a real (because that's easier).

Suppose then that we need to call modulus between the result and another true int.

So, for example:
var x = 3 ^ 2 ; //x = 9.0
x % 3 ;

If "x" is real (as in my example), then this throws an error since mod doesn't like doubles. Is there a way to safely convert "x" back to a true int for this operation? I've read that if you do "int y = x;" there's a chance that "x" will have been stored as a double truly slightly less than 9, so "y" will then be 8 due to truncation. Is this true, and if so, how can it be avoided?

You don't have to support modulus of reals.

The only thing you can safely do when converting a real to an int is to round or truncate. Not all real numbers can be represented in floating point exactly. Try printing:

printf("%.2f\n",0.3 + 0.6);