Recursion

Definition: Any time the body of a function contains a call to the function itself.

So, just as you are allowed to call function B from within function A, you are ALSO allowed to call function A from within function A!

Potential Problem: But if my function calls itself, how can we ever finish executing the original function?

What this means is that some calls to the function MUST NOT result in a recursive call. I think this can best be seen from an example.

```
// Pre-conditions: e is greater than or equal to 0.
// Post-conditions: returns b^e.
int Power(int base, int exponent) {

    if ( exponent == 0 )
        return 1;
    else
        return (base*Power(base, exponent-1));
}
```

To convince you that this works, let's look at an example:

Say we were trying to evaluate the expression

Power(5, 2)

Power(5,2) returns 5***Power(**5,2-1) = 5***Power(**5,1)

To evaluate this expression we must evaluate:

Power(5,1) returns 5***Power(**5, 0)

Finally, we have:

Power(5,0) **returns 1.**

Thus, Power(5,1) **returns 5*1, which is 5.**

Finally, Power(5,2) returns 5***Power(**5,2-1) = 5*5 = 25,
and we are done.

General Structure of Recursive Functions

What we can determine from the example above is that in general, when we have a problem, we want to break it down into chunks, where one of the chunks is a smaller version of the same problem.

(In the case of Power, we utilized the fact that $x^y = x * x^{y-1}$, and realized that x^{y-1} is in essence an easier version of the original problem.)

Eventually, this means that we break down our original problem enough that our sub-problem is quite easy to solve. At this point, rather than making another recursive call, directly return the answer, or complete the task at hand.

So, ideally, a general structure of a recursive function has a couple options:

1) Break down the problem further, into a smaller subproblem
2) OR, the problem is small enough on its own, solve it.