Ch 15 Recursion

Recursion is an elegant approach to solving some programming problems.

✦✦ Definitions ✦✦

- A **recursive method** is a method that calls itself
- A problem can be solved by **recursion** if it can be broken down into successively smaller problems that are identical to the overall problem.
- **Base case** - smallest problem solution where recursion isn't needed, used to end recursive methods
- **Recursive case** - the general recursive solution to a problem
- In mathematics, a **recurrence relation** is an equation that recursively defines a sequence, once one or more initial terms are given: each further term of the sequence is defined as a function of the preceding terms.

❋❋ Common Examples ❋❋

Factorial

n! = n * (n-1) * (n-2) … * 2 * 1

Recursive definition:

factorial(0) = 1 /* base case! */

factorial(n) = n * factorial(n-1)

Code:

```
private static int factorial( int n) {
   if( n == 0) {
      return 1;   // base case
   }
   else {
      return n * factorial( n-1);
   }
}
```

Fibonacci

The most famous recurrence relation of all:

$Fib_n = Fib_{n-1} + Fib_{n-2}$

$Fib_0 = 0$

$Fib_1 = 1$

Quick quiz: What is Fib_7?

Code:

```
private static int fib( int n) {
   if( n == 0) { return 0; }  // base case
   else if( n == 1) { return 1; }   // another base case
   else {
      return fib( n-1) + fib( n-2);
   }
}
```

Ackermann's Function

OK, maybe not so popular, but one of my favorites… the numbers get large, fast.

A(m, n) = n+1, if m=0
= A(m-1, 1), if m>0 and n=0
= A(m-1, A(m, n-1)), if m>0 and n>0

Ack is exercise #9 on page 952 of our text.
You can see the numbers in the series here: en.wikipedia.org/wiki/Ackermann_function

☆☆ Common approaches ☆☆

Arrays - When an array is part of a recursive method, then an index into the array is usually passed in as well, to define the recursive and base cases.

Strings - Similar to arrays, if strings are part of a recursive method, then an index (or two) into the string is a parameter as well.

Common errors

Not coding a base case - can lead to an infinite loop
Not reducing the problem with each recursive call - another infinite loop
Not reaching the base case - bug in your code

✧✧ And my two cents ✧✧

Any recursive algorithm can also be written without using recursion… using iteration. The advantage of recursion is the elegance of the solution and small size of your code. The disadvantage is that recursion is usually much slower than iteration.

- Can you write `factorial()` and `fib()` methods above without using recursion?
- How about `rangeSum()` on p 938 or `Circles` on page 939?

The point: You've "got it" if you can look at a problem and see an iterative *and* a recursive solution, if one exists.

Recursive methods where the recursion appears at the end of the method is commonly called **tail recursion**.