Recursive Functions

A recursive function is a function that is defined in terms of itself.

A common recursive function that you've probably encountered is the Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, and so on. That is, you get the next Fibonacci number by adding together the previous two. Mathematically, this is written as

$$f(N) = f(N-1) + f(N-2)$$

Try finding F(8). No doubt, you have the correct answer, because you see that F(0)=0 and F(1) = 1. To be formal about this, we need to also define when the recursion stops, called the base cases. In this case, F(0)=1, and F(1)=1. So, the correct definition of the Fibonacci numbers is:

$$f(N) = \begin{cases} 0 & \text{if N } = 0 \\ 1 & \text{if N } = 1 \\ f(\text{N - 1}) + f(\text{N - 2}) & \text{if } N > 1 \end{cases}$$

In computer science, we often use the term "recursion" to refer to a procedure (function, subroutine) whose implementation calls itself. Here is an implementation of the Fibonacci function:

```
int Fibonacci(int x) {
  if (x == 0) return 0;
  if (x == 1) return 1;
  return Fibonacci(x-1) + Fibonacci(x - 2);
}
```

Consider the factorial function, n!. It is often defined as 1 * 2* 3 … * N-1 * N, with 0! defined as have a value of 1. We can define this recursively as follows:

$$f(x) = \begin{cases} x * f(x-1) & \text{if } x > 0 \\ 1 & \text{if x} = 0 \end{cases}$$

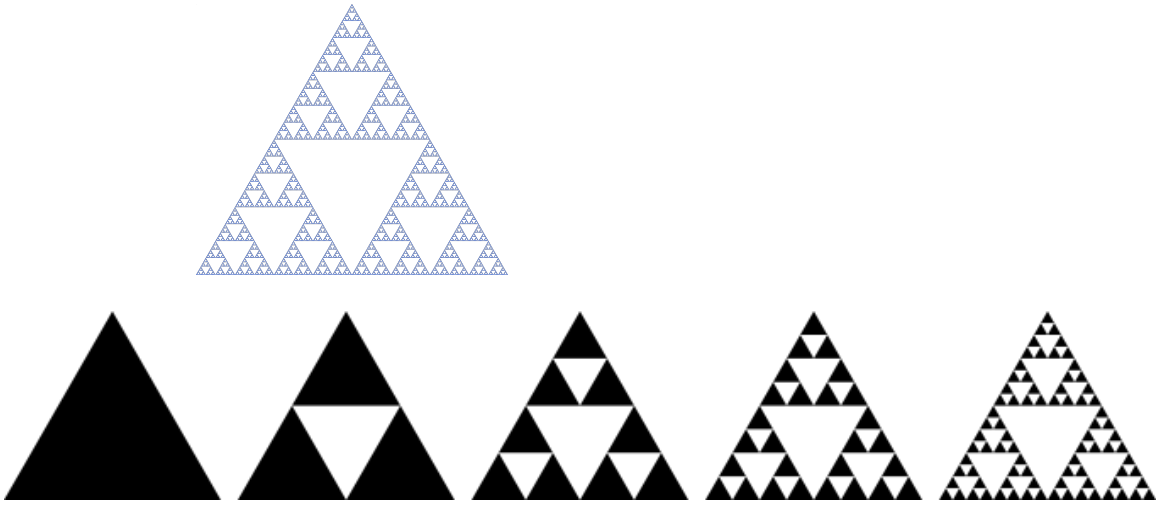
(We probably should say "x<=0" rather than "x=0" in the base case, to prevent the function from being undefined when x is a negative number.)

```
int Factorial(int x) {
  if (x<=0) return 1;
  return x*Factorial(x-1)(;
}
```

A few definitions: *Indirection recursion* is when a function calls another function which eventually calls the original function. For example, A calls B, and then, before function B exits, function A is called (either by B or by a function that B calls). Single recursion is

recursion with a single reference to itself, such as the factorial example above. And *multiple recursion*, illustrated by the Fibonacci numbers, is when a function has multiple self references.

The Sierpinski Triangle is an equalateral triangle, subdivided recursively into smaller equalateral triangles:



In computer science, recursion is an important technique for solving problems: One divides a problem into subproblems of the same type, solves each, and combines the results. The breaking down into smaller pieces happens until some "base case" is reached. You often hear this called "divide and conquer". To use that terminology, Fib(N) is solved by dividing it into Fib(N-1) and Fib(N-2), finding the results of each, and then combining the results by adding them together. Violla!

Many beginning programmers have a real tough time understanding recursion; so if this is confusing to you, not to worry!

In this ACSL category, we'll focus on mathematical recursive functions rather than programming procedures; but you'll see some of the latter, no doubt!

Find $f(11)$, given:

$$f(x) = \begin{cases} f(x-3)+1 & \text{if } x > 1 \\ 3x & \text{otherwise} \end{cases}$$

Find $g(9)$, given:

$$g(x) = \begin{cases} 1 & \text{when x is not positive} \\ g(x/2)+x & \text{when x is even} \\ g(x-1) & \text{when x is odd} \end{cases}$$

Find $g(13)$, given:

$$g(x) = \begin{cases} g(x/2)+g(x-1) & \text{when x is even and positive} \\ x & \text{otherwise} \end{cases}$$

Find $h(10)$, given:

$$h(x) = \begin{cases} h(h(x-5))+1 & \text{when x > 5} \\ x & \text{when 0 <= x <= 5} \\ 1 & \text{when x < 0} \end{cases}$$

Find $f(f(10,8), f(9,7))$, given:

$$f(x,y) = \begin{cases} f(x,y+2)+1 & \text{if } x > y \\ f(y+2,x-3)-4 & \text{if } x = y \\ 3x-y & \text{if } x < y \end{cases}$$