Recursion

Overview

Recursion is basically a function calling itself. It is often used when there is a pattern in the function. A good example of this is the Fibonacci sequence. In the Fibonacci sequence, the next term is the sum of the previous two terms (0, 1, 1, 2, 3, 5, 8 ...). If we make the function $f(n)$ return the nth Fibonacci number, then the pattern is $f(n) = f(n-1) + f(n-2)$. This is because the nth Fibonacci number is the sum of the $(n-1)th$ and $(n-2)th$ Fibonacci numbers.
There are two main parts to recursive functions, the base case and the recursive case. Since a recursive function calls itself, there needs to be a condition in which it stops calling itself or else it will go on forever, this is known as the **base case**. The recursive pattern that calls the function itself is called the **recursive case**.

Writing Recursive Functions

From our Fibonacci example above, we see that we have come up with the pattern, or recursive case. Now we need to write the base case. The base case is usually when the pattern doesn't make sense anymore. We can see that this starts when $n = 1$ because there isn't a Fibonacci number for $n = -1$. We know that the when $n = 0$ the Fibonacci number is 0 and when $n = 1$ the Fibonacci number is 1, so we can construct the following function for Fibonacci.

```
public int Fib(int n) {
    if (n <= 1) {
        return n;
    }
    return Fib(n - 1) + Fib(n - 2);
}
```

Here we can clearly see the split of the base case (if block) and the recursive case (rest of function). The base case usually comes first because we always want to stop, if we should, before making more recursive calls.

In general, I like to come up with the recursive case first and then try to fix a logical base case in for the recursive case that I came up with. This will usually help minimize how many recursive cases are written because the bulk of a recursive function is the recursive case.

Recursive Leap of Faith

The Recursive Leap of Faith is what we like to call the idea of trusting that the function does what it claims it does. In the Fibonacci example, we claim that the function Fib(int n), will return the nth Fibonacci number, even though it's the exact function we are currently writing. If we pretend that it always returns what it claims to return, we can use it when implementing Fib(int n), specifically for the recursive case. Sometimes you may run into problems when using the Recursive Leap of Faith in implementing a recursive function, but this is **NOT** because this idea is broken, but most likely because you forgot an important aspect of the function. Make sure you cover all aspects of the function while you are implementing it and the Recursive Leap of Faith will save you some confusion and time.

Solving Recursive Functions

The most important part of solving recursive functions is to **TRACE YOUR CODE CAREFULLY**. This means no cutting corners unless you are confident on a pattern and keeping track of all variables as well as which recursive frame you are in. Draw each frame (each time the function is called) and keep track of what each variable is during each frame.

It's okay to get lost when tracing, but make sure you practice and get comfortable with tracing. The more comfortable you get with tracing, the faster you become at it!