Chapter 21: Recursion

Section 21.1: Compute nth Fibonacci number

Visual Basic.NET, like most languages, permits recursion, a process by which a function calls *itself* under certain conditions.

Here is a basic function in Visual Basic .NET to compute Fibonacci numbers.

```
''' <summary>
''' Gets the n'th Fibonacci number
''' </summary>
''' <param name="n">The 1-indexed ordinal number of the Fibonacci sequence that you wish to receive.
Precondition: Must be greater than or equal to 1.</param>
''' <returns>The nth Fibonacci number. Throws an exception if a precondition is violated.</returns>
Public Shared Function Fibonacci(ByVal n as Integer) as Integer
    If n<1
        Throw New ArgumentOutOfRangeException("n must be greater than or equal to one.")
    End If
    If (n=1) or (n=2)
        ''Base case. The first two Fibonacci numbers (n=1 and n=2) are both 1, by definition.
        Return 1
    End If
    ''Recursive case.
    ''Get the two previous Fibonacci numbers via recursion, add them together, and return the result.
    Return Fibonacci(n-1) + Fibonacci(n-2)
End Function
```

This function works by first checking if the function has been called with the parameter `n` equal to `1` or `2`. By definition, the first two values in the Fibonacci sequence are 1 and 1, so no further computation is necessary to determine this. If `n` is greater than 2, we cannot look up the associated value as easily, but we know that any such Fibonacci number is equal to the sum of the prior two numbers, so we request those via *recursion* (calling our own Fibonacci function). Since successive recursive calls get called with smaller and smaller numbers via decrements of -1 and -2, we know that eventually they will reach numbers that are smaller than 2. Once those conditions (called *base cases*) are reached, the stack unwinds and we get our final result.