CmpSci 187 Discussion #6: Fibonacci Numbers Individual Handout

Marc Liberatore and John Ridgway

2 March 2015

Our goal today is to explore the behavior of a recursive method that computes a mathematical function, and write an alternate version that computes the function much more quickly.

The **Fibonacci function** is defined recursively for all non-negative integers n. We define $F(0)$ to be 0, $F(1)$ to be 1, and for any n with $n \geq 2$, $F(n) = F(n-1) + F(n-2)$. So $F(2) = F(1) + F(0) = 1 + 0 = 1$, $F(3) = F(2) + F(1) = 1 + 1 = 2$, $F(4) = F(3) + F(2) = 2 + 1 = 3$, $F(5) = 3 + 2 = 5$, $F(6) = 5 + 3 = 8$, and so on.

Just as with the factorial function presented in Lecture 9, we can use the recursive definition to write a recursive method to compute $F(n)$ for any non-negative integer n. Here is this method, embedded in a class whose main method allows us to run the recursive method on any input from the command line:

```java
public class Fibonacci {
  public static int fib(int n) {
    // computes fibonacci function for any non-negative integer n
    if (n <= 1)
      return n;
    return fib(n - 1) + fib(n - 2);
  }

  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Needs an integer parameter");
      System.exit(1);
    }

    int n = Integer.parseInt(args[0]);
    System.out.println(fib(n));
  }
}
```

Exercise 1: Use this program to find $F(10)$, $F(15)$, and $F(20)$.

Note: This program requires an input parameter, but we assume that you are running it from Eclipse rather than the terminal, so there is apparently no way to enter a parameter. Actually, there is:

- Right-click on the project `fibonacci-numbers`.
- In the menu that appears, choose Run As, and then Run Configurations. A window will pop up titled Run Configurations.
- Choose the Arguments tab.
- In the text box labeled Program arguments enter, e.g., 10 (or 15, or 20).

- Click on the Close button to close the window and save the argument, or on the Run button to close the window, save the argument, and run the program.

Exercise 2: Explain why this code is correct, using DJW's three questions for a recursive method

1. Does it have a base case, and is the output correct there?
2. Does it always make progress toward the base case, given its precondition?
3. If all recursive calls return the correct output, must it also return the correct output?

Exercise 3: For large enough n, this program will return an incorrect output due to integer overflow. Integer overflow can be detected when you get a negative output (if you add two large enough positive numbers it overflows and yields a negative result). Be careful to "sneak up" on the value that gives an overflow, because if you use a large enough number the result will become positive again; wrong but positive.

Try to exhibit this overflow behavior behavior, but don't let your program run for more than one minute. (When it has run for more than a minute, by the clock, simply click on the red square in the run area.) What is the largest n for which `Fibonacci` will compute $F(n)$ in less than one minute?

Exercise 4: Now write a new non-recursive `fastFib` method (in the Fibonacci class) that runs faster than `fib`. (One way to do this is to make an array for the previous values of `fastFib(i)`, and look up those values rather than recalculate them as `fib` does. Another is to remember only the last two values as you successively calculate $F(2)$, $F(3)$, ... , all the way to $F(n)$). You can modify the `main` method to test this method.

Exercise 5: With this new method, find the first n that causes integer overflow in the calculation, if you have not found it already.

Exercise 6: Write another new method `longFastFib`, similar to `fastFib` but that uses `long` variables in place of `int` variables to avoid the integer overflow problem. Can you demonstrate `long` overflow? If so, what is the first n for which this occurs?