

# CS 12 • Winter 2003

## In-lab Programming Exercise 2

### Recursion

wagner@cs.ucr.edu  
Department of Computer Science and Engineering  
University of California, Riverside

February 3, 2003

This second assignment is entirely devoted to recursion and is divided in four parts, all of which are mandatory. Remember that you must write an explanation section for all your work. In particular, you should use that section to answer the questions awaiting you below.

All four parts deal with the Fibonacci sequence. Fibonacci was a mathematician born in the Italian city of Pisa (where the famous Leaning Tower is) around the year 1175. His real name was Leonardo of Pisa but because his father's name was Guglielmo Bonaccio, Leonardo liked to call himself Fibonacci, short for *filius Bonacci*, which is latin for 'the son of Bonacci.' You can find more about him on the web, for example, [here](#).

As it turns out, the history behind his famous sequence is quite interesting. Apparently, Fibonacci wanted to know how fast rabbits can breed, so he made some simplifying assumptions and created a model to describe their mating behavior. You can find more about this fascinating aspect of his research [here](#). Our interest in the Fibonacci sequence stems from the fact that it can be described recursively:

$$F(n) = F(n - 1) + F(n - 2) \quad (n \geq 3).$$

Of course, we need starting values and the standard choices are  $F(1) = F(2) = 1$ . These give rise to the sequence  $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots\}$ . Although not obvious at first, it turns out that the Fibonacci numbers grow exponentially with  $n$ , that is, they grow *very* quickly. Your task in this exercise will be to explore some of the issues involved in computing the Fibonacci numbers.

## Part 1

(a) Write a program to **recursively** compute the Fibonacci numbers. You should first write a separate function `int FiboRec(const unsigned int n)` which returns the value of the  $n$ -th Fibonacci number. Once you have that, write your `main()` function to compute and print the first 50 Fibonacci numbers. Compute and print them one at a time; do **not** compute them all first. In other words, the output of your program should be similar to this:

$n$	$F(n)$
1	1
2	1
3	2
4	3
...	...
50	whatever it is

where each line is printed as its corresponding  $F(n)$  is computed. Be warned that it may take a while for the last several numbers to be computed. In particular, you may have to manually abort your program (I had to do so when trying to compute  $F(50)$ ). You should take a moment to appreciate how inefficient a recursive computation of the Fibonacci numbers is.

**Answer:** See the source code online.

(b) Once you have a table of the Fibonacci numbers computed recursively, take a closer look at it. Notice that some numbers are negative. How can that be considering that every Fibonacci number is strictly positive? You should try to figure that out by yourself without asking the TAs or your friends. There's an important lesson to be learned here and it's important that you find it out by yourself. If this is a bug of some kind, what would you suggest to fix it?

**Answer:** The reason the negative numbers appear is that the Fibonacci numbers grow very quickly outside the range of values that an `int` can hold. Recall, for example, that a *signed* 32-bit integer can be as large as — but no larger than —  $2^{31} = 2,147,483,648$ . So, as soon as the Fibonacci numbers you're trying to compute grow larger than that value, they will “wrap around” on the negative side. The solution, of course, is to use a numerical type that can hold larger values. Considering how fast the Fibonacci numbers grow, the best solution is to use `doubles` rather than `ints`.

(c) Now comment away the statements which correspond to the base case of the recursion and run your program. What happens? Why?

**Answer:** You should get a runtime error. Which one? A **stack overflow** error, of course. Why? Because, by removing the base cases, you're preventing the recursive function from ever stopping recursing. It's only when the stack overflows with all the activation frames being created by the recursive calls that the recursion stops. But, by then, it's too late... your program has crashed already.

## Part 2

(a) Now do exactly as you did in Part 1 above, but do so **iteratively**. If you did things the right way, you only need to change in your `main()` which function you're calling, `FiboRec()` or `Fibolter()`. Can you tell (subjectively) how much faster the iterative version is compared to the recursive one?

**Answer:** See the source code online.

(b) Once again, after you have a table of the Fibonacci numbers (now computed iteratively), take a closer look at them. Are there negative numbers still? Why?

**Answer:** Yes, there are still negative numbers showing up, because this problem has nothing to do with whether you're using recursion or iteration; it has to do with the range of the data type you're using to store the results of your calculations.

## Part 3

Using the functions you wrote for parts 1 and 2 above, write another program that measures how long each computation takes. In other words, the goal is to print a table like the following, where time is measured in clock units (an arbitrary amount which is computer dependent):

$n$	$F(n)$ iter	time (iter)	$F(n)$ rec	time (rec)
1	1	0	1	0
2	1	0	1	0
3	2	0	2	0
4	3	0	3	0
...	...	...	...	...
35	9227465	0	9227465	110
...	...	...	...	...
40	102334155	0	102334155	1196
...	...	...	...	...
50	...	...	...	...

For your convenience, I'm giving you the *entire* `main()` except for the two functions you need to implement. Am I a nice guy or what? There's a catch, however. You *still* need to explain what is happening in `main()` in your explanation section, and asking the TAs for help on that is a no-no. You're going to have to do some digging in the book to understand this code — that is intentional. By the way, as usual, free code can be downloaded from the course web page so you don't have to copy all this.

**Answer:** See the source code online.

## Part 4

In this part, I want you to write yet another program using your recursive function to compute the Fibonacci numbers but this time I want you to print the number of recursive calls made in the computation of a given Fibonacci number. For example, in the recursive computation of  $F(4)$ , we need  $F(3)$  and  $F(2)$ . But  $F(3)$  needs  $F(2)$  and  $F(1)$ , while  $F(2)$  is a base case, just as  $F(1)$  is. Thus, in order to compute  $F(4)$  recursively we need to make 4 calls: one  $F(3)$ , two  $F(2)$ 's and one  $F(1)$ .

Your program should output a table like this:

$n$	$F(n)$	num of rec calls
1	1	1
2	1	1
3	2	2
4	3	4
...	...	...
50	...	...

What can you say about how fast the number of recursive calls is growing as we increase the value of  $n$ ?

**Answer:** The easiest way is to define a variable visible outside of your recursive function and use it to keep a count of the number of recursive invocations:

```
\\ showing only the relevant parts...
```

```
int count_of_recursive_calls = 0;

int FiboRec(const unsigned int n)
{
    // it's as simple as adding this one line here !
    count_of_recursive_calls++;

    if (n < 3)
        return 1;
    else
        return FiboRec(n-1) + FiboRec(n-2);
}
```

As for how fast the number of recursive calls grows as we increase  $n$ , the answer is that it grows as a Fibonacci number as well!

■