Due date: Wednesday, Mar 13 (11:59 PM)

CS 271, Winter 2013 — Programming Assignment 4

Submission instructions

- Submit a single MIPS assembly file via TEACH: `http://engr.oregonstate.edu/teach`
- Your file must be named `PA4`[your-username]`.asm`
- For example, my submission would be named `PA4walkiner.asm`

Learning objectives

The goal of this assignment is to get more practice with algorithm design, procedure calls, and interacting with arrays in MIPS assembly language. You will also use recursion and function pointers for the first time. By the end of this assignment, you should be able to:

1. implement a recursive function in assembly
2. implement a *memoized* recursive function
3. pass function pointers as arguments and call functions passed in this way
4. adhere to procedure calling conventions

Description and requirements

In this assignment you return to the exciting domain of Fibonacci numbers! Specifically, you will be comparing the time performance of two different recursive functions for computing a Fibonacci number—one that is purely recursive, and one that uses a memoization table to quickly return previously computed results.

Your program will be organized into a number of procedures. You must define and use at least the following procedures in your program. (More is fine.)

void main()

- Print an introduction including your name, a title, and a notice of any extra credit you attempted.
- Call the procedure **getN** and save the result, n.
- Print a "purely recursive" banner, as in the example output.
- Call the procedure **testFib**, passing a pointer to **fib** and the value n.
- Print a "with memoization" banner, as in the example output.
- Call the procedure **testFib**, passing a pointer to **fibM** and the value n.
- Use the exit system call to end your program.

int getN()

- Prompt the user to enter a number between 1 and 25 (inclusive); read in the input (n).
- Check that n is within range. If not, print an error message and repeat until the user enters a valid number.
- Return n.

int fib(int *n*)

- Calculate and return the *n*th Fibonacci number using a *recursive* algorithm.

int fibM(int *n*)

- Calculate and return the *n*th Fibonacci number using a *memoized recursive* algorithm.

void testFib(f, *n*)

- This procedure takes two arguments: The first, f, is the *address* of a function that takes an integer and produces an integer (i.e. a function pointer). The second, *n*, is an integer.
- Get the current system time, *start*, using system call 30—you will have to look at the MARS documentation ("?" button) to learn the interface to this system call.
- Call the function f with argument *n* and save the result.
- Get the current system time, *stop*.
- Print out the result and the time it took to compute the value, calculated as *stop* − *start*.

Additional requirements

- Your code must respect the calling conventions in all non-leaf procedures. Refer to the stack diagram from the procedure call slides on the course web page.
- Your code must be fully documented. Documentation includes a header block (name, date, description), a procedure block for each procedure (pseudocode implementation, mapping of pseudocode variables to registers), and inline comments that relate the assembly to the pseudocode.

Helpful information

- You already implemented the **getN** function as Section 2 of the second programming assignment. You may be able to reuse (copy-and-paste) that code here.
- The recursive and memoized implementations of the factorial functions we went through in class will be very useful for you! They are posted on the class web page.
- I would strongly suggest getting **fib** and **fibM** working before writing **testFib**. This will make debugging your program significantly easier.

Extra credit (Small amount. Standard requirements must be fulfilled first!)

1. Implement **fibI**, a function that takes an integer and returns an integer, that computes Fibonacci numbers using an *iterative* algorithm. Test and time this function in the same way as **fib** and **fibM**.
2. Do something astoundingly creative! If you do, note it in the description you print out in the introduction, and make it clear that it might be extra-credit worthy. :)

Example execution

Below is an example execution that illustrates what your program should do. Your output need not match this output exactly. The important thing is that it must be clear and easy to tell that your program is computing and printing the correct results.

Program output is in `typewriter font` while user input is in ***red, bold, italic font***.

```
Computing Fibonacci numbers, Part II
    by, Eric Walkingshaw

Enter an integer in the range [1..25]: 0

That number was out of range, try again.
Enter an integer in the range [1..25]: 30

That number was out of range, try again.
Enter an integer in the range [1..25]: 17

== Purely Recursive ==
Result: 1597
Time: 1046

== With Memoization ==
Result: 1597
Time: 14
```