Assignment 6 — Functions

Due: Monday, February 28th

Like most high-level languages, C supports recursive functions. Since all high-level code must eventually be translated into low-level machine code, it must be possible to compile such functions into equivalent sequences of assembly instructions.

Deliverables:

Required Files: `main.c`, `fib.asm`

Optional Files: *none*

Part 1: Ground Rules

Throughout this class, any C code you write must compile using the command:

```
>$ gcc -Wall -Werror -ansi -pedantic ...
```

Furthermore, your C programs are expected to compile and run on Cal Poly's Unix servers.

Part 2: Fibonacci Numbers

The n^{th} *Fibonacci number*, denoted f_n, is given by:

$$f_n = f_{n-1} + f_{n-2}, \text{ where } f_0 = 0 \text{ and } f_1 = 1$$

For example, $f_0 = 0$ and $f_1 = 1$, thus, $f_2 = f_1 + f_0 = 1 + 0 = 1$. By the same reasoning, $f_3 = 1 + 1 = 2$, $f_4 = 2 + 1 = 3$, and $f_5 = 3 + 2 = 5$. The Fibonacci numbers are an example of a recursively defined sequence: each term in the sequence is defined using one or more of the preceding terms.

Because the Fibonacci numbers are recursively defined, they can easily be computed by a recursive function. Consider the C function given in `fib.c`: given a natural number n, it computes the n^{th} Fibonacci number[1].

Part 3: Functions in C

Complete the C program in `main.c`:

- Your program must prompt the user to type an integer, then print the corresponding Fibonacci number.
- Your program must use the function in `fib.c` to compute the printed Fibonacci number.
- You may add helper functions to `main.c` if desired, however, you may *not* alter `fib.c` or `fib.h`.

You may assume that the user will type an integer, then strike the "Enter" key. You may further assume that all integers encountered in this assignment will always fit into an ordinary, signed, 16-bit `int`[2].

For example:

```
>$ gcc -Wall -Werror -ansi -pedantic main.c fib.c
>$ ./a.out
Enter an integer: 6
f(6) = 8
```

[1] This is neither the most efficient nor the most elegant way to write this function. It has been intentionally written in such a way as to ease its eventual translation into LC-3 assembly.

[2] In practice, larger values will not be tested, as the runtime of `fib` is exponential.

Your program will be tested using `diff`, so its printed output must match *exactly*. Sample input files and their corresponding expected output files have been provided so that you can test this from the command prompt:

```
>$ gcc -Wall -Werror -ansi -pedantic main.c fib.c
>$ ./a.out < in1.txt > temp.txt
>$ diff temp.txt out1.txt
```

…if the files match exactly, then `diff` will output nothing.

Part 4: Functions in Assembly

Complete the LC-3 assembly file `fib.asm` by translating the given C function `fib` into the equivalent assembly function `FIBFN`:

- Do not attempt to optimize your function. Take a literal approach to translating the C function `fib` exactly as it was given.
- Your function must follow the LC-3 calling convention presented in lecture.
- You may *not* alter the existing function `MAINFN` in `main.asm`.
- It must be possible to rerun your program by manually resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized or that any files be reloaded.

Due to the limitations of basic LC-3 input and output, you may assume that the program as a whole will only be tested using integers between 0 and 6. However, your implementation of the assembly function `FIBFN` is expected to be able to handle the same range of integers as the C function `fib`.

For example:

```
Enter an integer: 6
f(6) = 8
```

Your program will be tested using `diff`, so its printed output must match *exactly*.

Develop this function with an idea of what the runtime stack should look like throughout its execution: draw the runtime stack for a simple test case such as $n = 3$ first, then consider the actual assembly code. Remember, for any non-trivial test case, some of the stack frames are going to be overwritten as the program runs, so it will be difficult to debug based only on the state of the runtime stack at termination.

Part 5: Submission

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `main.c` — A working C program to recursively compute Fibonacci numbers, as specified.
- `fib.asm` — A working LC-3 assembly function to recursively compute Fibonacci numbers, as specified.

The following files are optional:

- *none*

Any files other than these will be ignored.