Assignment 6 — Functions

Due: Wednesday, May 18th

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:

GitHub Classroom: `https://classroom.github.com/a/YOdnkRr2`

Required Files: `main.c`, `gcd.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: Euclid's Algorithm

The *greatest common divisor* of two integers a and b, denoted $\text{gcd}(a, b)$, is defined as the largest integer by which both a and b are divisible. If a and b are known to be positive integers, then:

$$\text{gcd}(a,b) = \begin{cases} a & a = b \\ \text{gcd}(a, b-a) & a < b \\ \text{gcd}(a-b, b) & a > b \end{cases}$$

For example, given $a = 6$ and $b = 8$, $\text{gcd}(6, 8) = \text{gcd}(6, 2) = \text{gcd}(4, 2) = \text{gcd}(2, 2) = 2$. Because the greatest common divisor is recursively defined, it can easily be computed by a recursive function. Consider the C function given in `gcd.c`: given positive integers a and b, it computes $\text{gcd}(a, b)$ recursively[1].

Part 3: Functions in C

Complete the C program in `main.c`:

- Your program must prompt the user to type two integers, then print their greatest common divisor.
- Your program must use the function in `gcd.c` to compute the printed greatest common divisor.
- You may add helper functions to `main.c` if desired, however, you may *not* alter `gcd.c` or `gcd.h`.

You may assume that the user will type two positive integers separated by a single space, then strike the "Enter" key. You may further assume that all integers encountered will always fit into an ordinary `int`.

For example:

```
>$ gcc -Wall -Werror -ansi -pedantic main.c gcd.c
>$ ./a.out
Enter two positive integers: 6 8
gcd(6, 8) = 2
```

[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.

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

```
>$ gcc -Wall -Werror -ansi -pedantic main.c gcd.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 `gcd.asm` by translating the given C function `gcd` into the equivalent assembly function `GCDFN`:

- Do not attempt to optimize your function. Take a literal approach to translating the C function `gcd` 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 in the range $\{1, \ldots, 9\}$. However, your implementation of the assembly function `GCDFN` is expected to be able to handle the same range of integers[2] as the C function `gcd`.

For example:

```
Enter two positive integers: 6 8
gcd(6, 8) = 2
```

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

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 compute greatest common divisors, as specified.
- `gcd.asm` — A working LC-3 assembly function to compute greatest common divisors, as specified.

The following files are optional:

- *none*

Any files other than these will be ignored.

[2] Larger arguments can be tested by setting breakpoints in `MAINFN` to pause execution, giving the opportunity to manually alter the arguments on the runtime stack as `GCDFN` is being called.