CS2630: Computer Organization
Homework 1
Practicing C programming

In this homework, you will improve these skills:

- Write a C program involving printf, loops, if-statements, functions
- Write a C function that reads and/or manipulates arrays

Preparation

You should have completed the lab assignments, readings, and lectures dealing with pointers and arrays in C.

Coding requirements

- Your C program must compile with **no errors and no warnings**, using this command
    - gcc -Wall -o filename filename.c
- After compiling, your program should run using this command
    - ./filename
- For Problem 1 and Problem 2, you should not put all the code in the main function; keep main() short and have it call other functions instead
- Include at least the following comments
    - Comment **above** each function saying ***what*** it does (not ***how*** just ***what***), particularly what the arguments and return value mean.
    - Comment at top of the file saying 1. Your name, 2. Semester, 3. what the application does
- Use appropriate variable names
    - Longer names for important variables that are used in a larger scope
    - Shorter names for less interesting variables, such as looping variables, or that are used in a smaller scope
- Your program ***must be portable***. You can typically achieve portability by: 1) not having any compiler warnings (see above) AND 2) not using any libraries other than the C standard library. We will grade the homework using the ***CLAS Linux machines*** (i.e., fastx), so we recommend you compile and run your program at least once on them. Alternatively to fastx, making sure it works on repl.it is fine.
- OPTIONAL: Make your code look nicer. Often coding style is enforced by your collaborators when they review your code. However, since this is a solo project, we suggest running the following beautifier on your code. It will fix spacing, indentation, and other basic style issues.

```
clang-format -i filename.c
```

Problem 1: Reverse Fibonacci

Create a new file called reversefib15.c. When compiled and run, it should print the first 15 fibonacci numbers ***from large to small***. One number per line.

You will submit your working reversefib15.c file.

Problem 2: Reverse Fibonacci with an argument

Copy your solution to reversefib15.c to a new file reversefib.c. When compiled and run, it should print the first N fibonacci numbers ***from large to small***. One number per line.

You will run your program with this command
```
./reversefib N
```
Where N is some positive integer, for example
```
./reversefib 17
./reversefib 25
```

For an example of how to take an integer argument on the command line, refer to arguments.c. You may use that code and modify to suit your need.

You will submit your working reversefib.c file.

Some computer organization fun: run reversefib on a larger value of N (e.g., 50 or higher). What do you notice? What do you think is causing this behavior?

Problem 3: Move Zeros to End of Array

Complete the zero_last function in zerolast.c. The function takes an array and its length as arguments and changes the array so that it has all its zeroes at the end. For example,

```
[-1,-2,-2,4] => [-1,-2,-2,-4]
[0,4,3,2,1] => [4,3,2,1,0]
[5, -1, 10, 0, 11, 0] => [5,-1,10,11,0,0]
[0, 0, -4, 5, -6, 7, 0, 4, 3] => [-4,5,-6,7,4,3,0,0,0] [1,0,0] => [1,0,0]
```

You will submit your working zerolast.c file.

Problem 4: Max integer found in two arrays

Complete the max_both function in maxboth.c. The function takes as arguments two arrays (and their lengths) containing only positive integers and returns the maximum integer found in both arrays. If no duplicates are found it returns -1. For example:

> [2,4,6,8,10,12,14] [7,7,5,4] => 4
> Returned 4 because both arrays have 4. Other numbers are bigger but are not in *both* arrays.
>
> [2,4,6,8] [3,5,7] => -1
>
> Returned -1 because there are no values found in both arrays
>
> [7,6,22,4,4] [4,5,5,25,6] => 6

You will submit your working maxboth.c file.

Reflection report

Turn in a typed document with answers to these questions.

1. Explain how zero_last manages to have an effect despite its return value being void.
2. In your code for zero_last and max_both, did you treat the parameters like arrays, like pointers, or like both? Explain with examples. Could you have kept the algorithm the same but used different C syntax to achieve the same effect? Give at least one specific example.
3. Tell a debugging story
    a. Observation: describe a time, while working on one of these programs, where you observed an undesired behavior. What was the undesired behavior and what was the expected behavior?
    b. Investigation: for the situation above, describe what you specifically did to look for the cause.
    c. Test: what fix did you attempt? What was the result when you tested it?

If you get a segmentation fault when you run your program

(Relevant to Problem 3 and Problem 4, which use arrays). A segmentation fault indicates a memory error; in the case of these two programs, it is most likely that you are trying to access an array beyond its bounds. So that might be the first thing to look for in your code.

In the later programs you work on, the memory errors will be more involved, requiring debugging tools to diagnose and fix. You will learn a methodology for using these tools in prelab/lab2.