C Intro / Refresher Lab # 1

Golden G. Richard III

DO NOT TURN ANYTHING IN! FEEL FREE TO WORK WITH OTHERS ON THIS LAB.

The intention of this lab is to brush some of the rust off your C skills. If you have little or no familiarity with C, work with someone who does. Raise your hand if you need me to set up a partnership for you. If your skills are shiny, do it anyway and help someone near you understand what's going on.

Why?

Many reasons. Here are some:

- Java doesn't make you strong.
- C makes you strong.
- Python is really expressive, but writing everything in Python is a really bad idea.
- Operating systems are written in C. This won't change anytime soon.
- Many, many important applications are written in C or C++.
- If you're interested in cybersecurity, both malware and exploits are commonly written in C or C++.

(1)

Write a simple C program that prompts a user for an integer *n*, then accepts *n* strings from standard input (one per line). Use `fgets()` to read each string. If the input is longer than 1024 characters, truncate it to 1024 characters. Remove the newline from the string that `fgets()` appends, unless truncation already removed it. All of the strings should be stored in a dynamically allocated array (with exactly *n* elements), with each element containing exactly the right number of characters to hold the string and the terminating null character. Once input is complete, use the standard library function `qsort()` to sort the strings and then output the sorted list using `printf()`.

Use the following command under Linux to compile your program (assuming you named it `prog1.c`):

```
$ gcc -Wall -o prog1 prog1.c
```

You should receive no warnings!

(2)

Consider the following type definition:

```
typedef struct funcs {
      int (*openit)(char *name, int prot);
      void (*closeit)(void);
} funcs;
```

(a)

Write a C program that includes simple C functions `my_openit()` and `my_closeit()` that match the types of the function pointers in the structure definition above. The functions don't have to do anything complex—inserting a single `printf()` statement in the body is sufficient. You should also write function prototypes for your functions.

(b)

Now declare a variable of type `funcs` and statically initialize the fields `openit` and `closeit` with the addresses of your open and close functions.

(c)

Now illustrate the initialization of the fields of a variable of type `funcs` using a C function `f()`, e.g., using `f(&var_of_type_funcs)`.

(3)

The program `blarg.c` "should" output the elements of the statically declared array, but it doesn't. Why?

(4)

Consider `blarg2.c`. What is going on?

(5)

Now consider `blarg3.c`. There's a mistake in here that even operating systems developers make. Why doesn't the program work correctly? Understand first, then fix.