LOUISIANA STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CSC 4103: Operating Systems
Prof. Golden G. Richard III

Programming Assignment # 0: C Refresher
Due Date: September 7, 2022 @ Class Time
NO LATE SUBMISSIONS

Unlike the other assignments in 4103, collaboration is explicitly allowed on this assignment, if everyone working together ends up understanding everything!

Note: Most people enjoy working on (5) alone, because it's "competitive".

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, then help someone near you understand what's going on.

Why C? Why? Why?

Many reasons. Here are some:

- Java doesn't make you strong.
- C makes you strong.
- Python is expressive and relatively easy to program in 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 C program **`prog1.c`** 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 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. `malloc()` should be used to create the array of strings. Once the input is processed, 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:

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

You should receive no warnings!

If you need a starting point, I've provided the program **`sort-ints.c`** on Moodle, which reads a set of integers and sorts them using `qsort()`. Make a copy of that program, name it **`prog1.c`**, and go from there.

---

(2)

Consider the following type definition:

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

(a)

Write a C program called **`prog2.c`** 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)`.

Use the following command under Linux to compile your program:

```
$ gcc -Wall -o prog2 prog2.c
```

You should receive no warnings!

---

(3)

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

---

(4)

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 it.

---

(5)

Now you get to write some truly awful looking C.

Consider the following output from a C program. The lines with the leftmost X's are against the left margin of the screen, the other lines begin with a single space, and there are single spaces between the X's:

```
  X  X  X  X  X
X  X  X  X  X
  X  X  X  X  X
X  X  X  X  X
  X  X  X  X  X
```

First, write any program in C that outputs the pattern above (exactly).

Now things get more interesting. Modify your program to satisfy the following rules:

Rule # 1: Your program should be as small as possible (in terms of the number of characters in the source file). Optimizing for the smallest possible source code size isn't generally good programming practice, but it can help you think about how to use language features in clever ways and deepen your understanding of those features.

Rule # 2: You cannot use a C statement that outputs more than a single character at a time. Hint: That means you're going to be using `putchar()`. ☺

Rule # 3: Your entire source code must reside in a single file called `small.c`. Your final solution can either not use header files at all or ONLY use standard C header files. You may not have custom header files.

Rule # 4: All code must really be in the single source file. You may not pipe code into `gcc`, etc., although we will look at this technique in class, because it's cool. ☺

For this assignment only, warnings during compilation are OK. Your program must simply work and be small. Very small.

Evaluate your solution under Linux as follows:

```
$ wc -c small.c                  // to reveal # of characters
$ gcc -o small small.c           // compile
$ ./small                        // execute
```

Smaller than 75 characters is good. Smaller than 50 characters is world class.

SUBMISSION INSTRUCTIONS:

- **Name your programs properly so the grader doesn't lose their mind. You *must* use these source file names:**
    - The solution to (1) should be called **`prog1.c`** and reside in a single source file.
    - The solution to (2) should be called **`prog2.c`** and reside in a single source file.
    - For (3), simply modify **`blarg.c`** and don't change the name of the source file.
    - For (4), simply modify **`blarg3.c`** and don't change the name of the source file.
    - For (5), name the source file **`small.c`**.

- **Put all of the source files into a directory called `prog0` in your 4103 account on classes.csc.lsu.edu.**

- **Use the following command to turn in your solution:**

```
$ ~cs4103_ric/bin/p_copy 0
```