ECE 2220: System Programming Concepts – MP1

The goal of this machine problem is to review C programming. This assignment provides a simple experience with which the user become familiar with:

1. Prompting for and processing text input from the terminal;
2. Manipulation of arrays; and
3. Utilization of conditionals and loops.

The interactive approach for prompting the user for data, reading input from the terminal and printing to the terminal will be utilized in many of the programming assignments this semester. **It is critical to adhere precisely to the format of the input and output** because we will utilize automatic tools to test for proper operation of your code. A template is provided that contains some of the formatted print statements. **You must use the template and these printf() statements. Do not change their text, though you may add additional print statements.**

Background: Our special agents use secret messages to communicate their status to Headquarters. Here at HQ, we de-crypt the messages sent by the agents. The decryption algorithm works as follows:

1. A *message* consists of a sequence of numbers between 10 and 100 ($10 < c < 100$), e.g., "23, 54, 38, 49, 76, 13".
2. All the messages are of the same size *msg_size*, where 6 <= *msg_size* < 12.
3. To decode a message, we need a predefined *key*, where 1 < *key* < 5.
4. A number is considered as a *signal* if it is equal to the sum of a *prime number* and the *key*.
5. Let *n* denote the number of *signals* contained in a message, *n* indicates the true meaning of the message:
    a. $n = 1$→ "I'm safe, all good."
    b. $n = 2$→ "Mission Success. Agent ID:"
    c. $n = 3$→ "Don't contact me."
    d. $n = 4$ → "Mission Failed. Agent ID:"
    e. All the other values → "No content."
6. In messages conveying mission status (i.e., $n = 2$ or 4), the last number in the original message is always the *Agent ID*.
7. As soon as a predefined *alert* number (-2) is read in, the program will exit immediately after printing "Agent center is not safe now. Go find a safe place. Program exit."
8. If a message contains any number that is out of the given range (less than 11 or greater than 99), the message is considered *corrupted.*

Example: Assume *msg_size* = 8, *key*=2, given a message of

"11(=9+2), 36(=34+2), 46(=44+2), 21(=19+2), 13(=11+2), 56(=54+2), 37(=35+2), 18(=16+2)"

The total number of *signal*s is 2. Therefore, the true meaning of the message is “Mission Success. Agent ID:”

Notes:

1. Use #defines for the constants. Never hard code parameters into your program.
2. Here are examples of the input a user might type (the format is critical), and the output that must be printed. The user types the numbers in the left column. Of course, your program will print additional output information so that the user is prompted for input values. Your prompts should be designed to make it easy for a user to input the values. We will test with extensive combinations of inputs that will try to break your program.

   Sample Input and Output:
   >>Minimum Required Output for the Sample Input:

   6 *(the size of each subsequent message)*
   2 *(the key to be used for all messages)*
   11 *(first message begins here)*
   12
   13
   14
   15
   16

   >>Mission Success. Agent ID: 16.

   13
   13
   13
   12
   14
   18

   >>Don't contact me.

   22
   23
   25
   26
   22
   23

   >>I'm safe, all good.

   8
   12
   13
   14

15
16

>>Corrupted Message.

-2

>> Agent Center is not safe now. Go find a safe place. Program exit.

3. You compile your code using:

   $ *gcc -Wall -g lab1.c -o lab1*

   The code you submit must compile using the -Wall flag and **no compiler errors or warnings** should be printed. To receive credit for this assignment your code must compile and at a minimum evaluate the set of samples given above. Code that does not compile or crashes before evaluating the first set will not be accepted or graded. Code that correctly detects the examples above but otherwise fails many other cases will be scored at 50% or less. All code must compile with no warnings and run correctly on the CES Ubuntu machines.

   While your program is designed to be interactive, to verify that your program can read input correctly download the example input file from Canvas and run your program as follows:

   $ ./lab1 < test_input

   The output may not be as nice as when you run interactively, but the output must contain the strings shown in the example above. (We will search for these specific lines. Note there are four sets in this example.)

   ***Note** that, your program must handle unexpected input, or 10 points off.

4. Submit a zip archive to Canvas.
5. Turn in a paper copy of all code files at the start of the first-class meeting following your submission. Print in landscape mode, two columns, with a small monospaced font size to save paper.

See the ECE 2220 Programming Guide for additional requirements that apply to all programming assignments.

Work must be completed by each individual student. See the course syllabus for additional policies.