CPE 325: Embedded Systems Laboratory
Laboratory Assignment #1

Assignment [50 pts]

1. **[25 pts]** You are working as a government cryptographer and are tasked with writing a C program that implements a Caesar cipher to encrypt a string. The encryption scheme should take any string of characters and shift the ASCII value of alphabetical characters right by 3. Numbers, spaces, and special characters should be left alone. For example, if the given string is "**Hello all, welcome to CPE325 Fall 2025!**", the output should be like the following:

   Khoor doo, zhofrph wr FSH325 Idoo 2025!

   Note: You can hard code the given string as follows, you may be asked to change it during demonstration:

   char msg[] = "Hello all, welcome to CPE325 Fall 2025!"

   To print the string in a single line using printf, the following statement

   can be used: printf("%s\n", msg);

   Hint: The ascii code equivalent for 'a' is 0x61

   More about the cipher: https://en.wikipedia.org/wiki/Caesar_cipher

2. **[25 pts]** You have been given a program Lab01_P2.c that computes the GCD of two hardcoded integers using the Euclidean algorithm. This program does not work in its current state: it has syntax errors (this will cause the project build to fail) and logical errors (once built & run, the program output is incorrect, or program behaves atypically). You are free to make any changes necessary to correct this given program, provided the correct output will still be given if the input value is changed. An example of correct program output is shown below:

   Input values: 138, 126
   Output GCD: 6

   Please also measure the number of clock cycles taken by the function call get_gcd(int a, int b) for at least two different pair of input values, and include these measurements in your report. How does the input value affect the

number of cycles taken?

Hints to get you started on debugging:

- **Check Your Output**: After running your code with the provided input values, calculate the GCD of these values manually using the Euclidean algorithm to ensure your output matches.
- **Syntax Errors**: Syntax errors will prevent your code from compiling and are indicated in the compiler's error messages. Correct the first error listed and recompile your code to check for additional errors.
- **Logical Errors**: To identify and understand logical errors, you might consider:
    - **Testing with Different Inputs**: Use a variety of pairs of integers as inputs (e.g., pairs of prime numbers, one large and one small number) and verify if the output is consistently correct.
    - **Break Down the Problem**: Determine whether any incorrect output is due to errors in how values are calculated or merely in how they are printed.
    - **Use a Debugger**: Step through your code with a debugger, examining the values of a and b throughout the execution of the get_gcd function. Ensure the loop and the modulus operation behave as expected.

Topics for Theory

1. Discuss the following tools/features from Code Composer Studio in your report
    a. Memory window
    b. Console window
    c. Variable window
    d. Breakpoints
2. What commands in Code Composer Studio can you use to run through your program?

Deliverables

1. Lab report which includes:
    a. Brief theory discussion
    b. A neat **flowchart** for each of the programs
    c. **Output screenshots** for both of the programs
    d. Source code (.c files) in appendix, included in lab report

Notes:

1. You must create an organized directory, subdirectory, workspace, and project

for each demo code and each solution.

2. During demonstration, you should be able to inspect variables, set watchpoints, set and monitor breakpoints, monitor registers and memory, and show the output.
3. While comparing the results for part 2, make sure you test different input sets.
4. The report (PDF) should be a single submission with the source code pasted at the end of it.