**COSC647**
**Homework 1 - Due Oct 5th, 2015.**

Acknowledge any help and do not google for solution

## Submission Instruction

**Failure to follow the below instructions will result in a 20% penalty.**

1. Do each programming problem in a single file with the name format: ***"<your_last_name>_HW_XX_YY.<extension>"*** where ***XX*** is the assignment number, ***YY*** is the problem number, and <extension> is ***.c*** or ***.cpp***. For instance, for assignment 01 and student name "David Smith", the filenames would be "Smith_HW_01_01.c", "Smith_HW_01_02.c", etc for questions #1, #2, and so on.

2. When you are done, create a folder named ***"<first_name>_<last_name>_HW_XX"*** and put all your source files (and only **source** files) in this folder. For example: folder name *"David_Smith_HW_01"* for student name "David Smith". No *other object* files or test files should be included.

3. Finally, create a ZIP file of this folder with the same name (i.e., "David_Smith_HW_01.zip"), change the file extension to ***.txt*** (i.e., change "David_Smith_HW_01.zip" to "David_Smith_HW_01.txt") and submit it over Blackboard.

1. Write a C program which takes into account two integers A, B and output the greatest common divisor of both A and B. Example: A = 10, B = 15 then output 5; A = 7, B = 19 then output 1.

2. Write a C program which asks the user for an integer N. Output the prime number P that is closet to N and is greater than N. Example: N = 10 then output P = 11. N = 23 then output P = 29.

3. Write a C program that allows the user to enter two integers and output their sum and product. Observe the behavior of your program when the two input integers were big.

4. Write a "`Hello World`" C program with some extra initialized and uninitialized variables. This program will also be used in questions 7, 8 9 and 10.
   When you compile it using `gcc`
   (a) What is the effect of the option `-Wall`?
   (b) What is the effect of the option `-pedantic`?
   (c) What is the difference in the size of the code compiled with and without `-g`?
   (d) What is the difference in the size of the code compiled with `-O1, -O2`, and `-O3`?

7. Open the C program in the debugger `gdb`. Set a breakpoint at the first line. Run the program to that point.
   (a) What is the current **`ESP`**?
   (b) What is the current **`EBP`**?
   (c) What is the current **`EIP`**?
   (d) Use the **`x`** command to show the assembly language code for the next few commands to be executed.

5. Use the **`strace`** command to list all of the system calls that your program makes.

9. Use **`objdump`** to find the address of the following sections in your program:
   (a) **`.text`**
   (b) **`.bss`**
   (c) **`.data`**

10. Rewrite the "`helloASM.asm`" program described in class. Compile it and link it.
    (a) Include the source, object, and executable code.
    (b) Debug the program. What memory addresses contain the program's environment strings?
    (c) Debug the program. What memory address contains the program's arguments?
    (d) Debug the program, and stop it immediately before the syscall for **`write`**.
        - What is the current **`ESP`**?
        - What is the current **`EBP`**?
        - What is the current **`EIP`**?
    (e) Debug the program. Where is the message string located?
    (f) What is the return value for the **`write`** syscall? Where is it located?

11. Write a C program that contains a stack based buffer overflow. Explain in detail why the program has a stack based buffer overflow flaw. Demonstrate the flaw by causing the program to crash with a segmentation fault. Include the state of the stack before the crash, and determine exactly why the program crashed.

12. Write a C program that contains a stack based buffer overflow, and make it SUID root. Run the program outside of the debugger as an unprivileged user, and exploit the overflow to obtain a root shell. Include a description of exactly how the program was exploited, and a screen shot showing that a root shell was obtained.