10. **Due:** Write a C program that contains a function call. [The program is otherwise arbitrary.]

   (a) Compile and run the program.
   (b) Use `gcc` to obtain the assembly language source of the program.
   (c) Compile the assembly language program, and run it. Check that it runs in the same fashion as the original.
   (d) Debug the program, and stop the program before the function call.
      i. What is the program’s EIP?
      ii. What is the program’s ESP?
      iii. What is the program’s EBP?
      iv. What is the current state of the program stack?
      v. Identify all of the local variables for the function `main()` on the stack. Verify that they hold the correct values.
   (e) Debug the program, and stop the program immediately after the function call.
      i. What is the program’s EIP?
      ii. What is the program’s ESP?
      iii. What is the program’s EBP?
      iv. What is the current state of the program stack?
      v. Identify all of the local variables for the function on the stack. Verify that they hold the correct values.
      vi. Verify that the correct return address for the function has been placed on the stack.

11. **Due:** 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. **Due:** Write a C program that contains a stack based buffer overflow, and make it SUID root. Run the program in 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. [Start/Utilities/Desktop/KSnapshot]