CPS 5401 Intro to Computational Science Name ______________________________
Fall 2012 Shirley Moore, Instructor
C Exercises
Due September 25, 2012

Turn in a gzipped tar file named <your name>-c-lab.tar.gz that contains your program source code and any input or output files for your solutions to the problems below by emailing the file to the instructor. You may turn in your answers to the questions below either electronically (included in your tar file) or on paper. Your tar file should unpack into a directory named <your name>-c-lab with subdirectories for the different problems. Do not include object or executable files in your tar file. The instructor should be able to build your programs by using the gcc command for a single file or your makefile for a solution that uses multiple files.

1. Write a C program containing a function that returns the standard deviation from the mean of an array of real values input from the user. Note that if the mean of a sequence of values ***(x_i, i = 1, n)*** is denoted by *m* then the standard deviation, *s*, is defined as:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - m)^2}{n}}$$

You may make use of additional functions if you wish.

To demonstrate correctness compute the standard deviation of the following numbers (10 of them):

```
5.0 3.0 17.0 -7.56 78.1 99.99 0.8 11.7 33.8 29.6
```

and also of the following 14,

```
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0
14.0
```

and redirect your output for both of the above to a file.

2. **Code profiling and registers.** In this problem, we will use some basic code profiling to examine the effects of explicitly declaring variables as registers. Please turn in the version of the code that runs the fastest. Consider the fibonacci sequence generating function fibonacci() in prob2.c which can be downloaded from the course website). The main() function handles the code profiling, calling fibonacci() many times and measuring the average processor time.

(a) First, to get a baseline (without any explicitly declared registers), compile and run prob2.c. Code profiling is one of the rare cases where using a debugger like gdb is discouraged, because the debugger's overhead can impact the execution time. Also, we want to turn off compiler optimization. Please use the following commands to compile and run the program:

```
$ gcc -O0 -Wall prob2.c -o prob2
$ ./prob2
```

How long does a single iteration take to execute (on average)?

(b) Now, modify the fibonacci() function by making the variables a, b, and c register variables. Recompile and run the code. How long does a single iteration take now, on average?

(c) Modify the fibonacci() function one more time by making the variable n also a register variable. Recompile and run the code once more. How long does a single iteration take with all four variables as register variables?

(d) Comment on your observed results. What can you conclude about using registers in your code?

3. Rewrite the fibonacci() function from problem 2 to use the loop constructs below. Make sure that your code is equivalent to the original function and will work for all possible positive values of NMAX.

(a) Use a while loop. Turn in your solution as prob3a.c
(b) Use a do…while loop. Turn in your solution as prob3b.c.

4. Write a C program that will use a (pseudo) random number generator to fill an array with 1000 unsigned integers between the values of 0 and the maximum possible value. Sort the array using the qsort() standard C library function. Then go into a loop that asks the user to enter a non-negative integer for which to search, uses the bsearch() standard C library function to perform the search, and outputs either the position in the array where the integer was found or "not found" if the integer is not in the array. The program should exit if the user enters a negative integer.

5. Finish the program to solve the linear Diophantine equation

$ax + by = c$, a and b non-negative integers

that was discussed in Practical Programming in C, Lecture 3. Your solution should consist of a header file euclid.h, a file euclid.c containing the gcd() and extended_gcd() functions, and a file diophant.c containing your main program. Your main program should input a, b, and c from the command line, check for correct input, and output either "No solution" if no solution exists or the solution. For example:

```
$ ./diophant 4 6 8
Solution is x=-4, y=4
```

Create a makefile for your program that will rebuild it correctly as needed.