ECE199JL: Introduction to Computer Engineering **Fall 2012**
Discussion Session 4 **Tuesday, 18 September**

Programming Practice

Log in to an Engineering WorkStation (EWS) machine, and copy the files

```
cp -r /class/ece199/disc4 .
```

into your home directory. `cd disc4` and list (`ls`) the files; you will find copies of slightly modified versions of the example program `dec2bin.c` and of another program `avg.c`, which computes the maximum and average of a series of integer inputs.

Part 1

Compile the program `dec2bin.c` using the command

```
gcc -Wall -o dec2bin dec2bin.c
```

Note that the `-Wall` flag tells the `gcc` compiler to display all warnings. Warnings inform the programmer of code constructions that are not illegal, but which are unusual and more than likely represent an action or error that the programmer did not intend. We strongly recommend that you compile with warnings enabled when you are developing codes, and that you debug or edit your code to remove all warnings. In our experience, this will catch many bugs or errors and greatly reduce the time it takes for you to develop working, bug-free codes!

Run and test the program (by typing `./dec2bin`). Enter the number 15, for which you know the corresponding binary representation, and confirm that the answer is correct. Continue to test the program by entering several other positive integers for which you can easily calculate the corresponding binary representation, and confirm that it always produces the correct answer.

Edit the program using your favorite Unix text editor (for example, `vim` or `gedit`; review the Lab 0 handout at the course website if you have forgotten). Study the first part of this program, which reads in the digits (in ASCII) of the number you enter and converts it into binary integer form stored in the variable `int number`. Once you think that you understand how this section of the program works, by using your preferred editor change the directive `#define TEST_FLAG_A 0` to `#define TEST_FLAG_A 1`. Doing so will activate certain `printf` statements that print out intermediate values of variables during each execution of the loop. Recompile the code and run it again.

Enter the number 4703. Calculate by hand or in your head what the values of `thischar`, `thisdigitval`, and `number` will be each time the program reaches the `printf` statement labeled CHECK A. Now compare those values with the ones produced by the program, and confirm that the program, and/or your "tracing" of the program execution, is correct. (NOTE: this process of tracing the program execution to confirm that it is performing exactly the operations you intended is your primary method of debugging your algorithms and codes. MAKE A HABIT of inserting test instructions in your codes for debugging purposes!)

Once you are sure you understand exactly what this first part of the program is doing and how it works, "uncomment" the statement `/*thisdigitval = thischar - '0';*/` and inactivate ("comment out") the following statement `thisdigitval = (int) (thischar & ascii_mask);`. Recompile the code and run this alternate version to confirm that it also produces the correct answer. Study both approaches and make sure you understand why they both successfully convert a digit from ASCII to binary representation. You will likely wish to consult an ASCII table from your textbook or the web in doing so.

Part 2

Now that you have fully analyzed and tested the decimal-to-integer part of `dec2bin.c`, restore `TEST_FLAG_A` to 0 and activate `TEST_FLAG_B`. Doing so will cause the program to print out the intermediate variables as the program reads out and prints the binary digits of the number. Once again, analyze the code mentally until you think you understand how and why the program works. Trace through the latter portion of the program manually, and confirm that your analysis agrees with the printed test values.

Part 3 (If you have time)

Once you have completed your analysis of `dec2bin.c`, move on to the program `avg.c`. The `avg.c` program finds the average value of a series of input integers.

Read through the code to understand how it works. It may be helpful to sketch out a flowgraph of the code. Add test `printf`s to the code to print out intermediate values to confirm your understanding and predictions. Compile and run the code to compute the average of the numbers -12, 10, 2, 3, -1, and 4.

Now alter the program to compute the average of all *positive* input values. Recompile, test, and debug your program until it works correctly; manually tracing your program's operation and printing out intermediate values of internal variables to confirm that the code is actually doing what you want it to is usually the fastest way to find errors.