This assignment is due by the start of class on Wednesday, Sept. 7. Kill some bits, save a tree – and save me having to decipher everyone's handwriting. Please type up your solutions and email them to me.

The source code for this assignment can be found in ~bwall/CS440/assign1.tgz. Extract the source into your own directory, compile and link each of the source files using *gcc*, and run the resulting executable. Each one will either crash and burn with the ever-so-helpful Unix *segmentation fault* or will generate some unexpected results.

Problems 1 through 10: for each of the *err*.c* programs, figure out what is causing the error in the program. Fix it in some way and make sure that the program will run successfully. Record what was wrong with the code, why it manifested itself the way it did (i.e. what did the error cause to happen in the code), and what you did to repair it.

Each question is worth three points – so you can get partial credit for each one. For full credit, make sure you are fairly explicit in explaining why the error programming error caused the erroneous results; I'd like to see how well people understand what's going on in the machine while the program is running.

Bonus questions. Feel free to skip these; if you have the time or inclination, they are worth bonus points that can carry over between assignments. (So if you get everything right on this assignment and get all the bonus points, you can get more than 100% for the assignment and offset some mistakes on another assignment.)

1. Figure out what's wrong with program *err_bonus1.c*. Be detailed when explaining why exactly the incorrect value was generated.

2. In program *err6.c*, you probably found and fixed a common C programming error and made the program run correctly. But it was a less easy to find programming error that actually caused the segmentation fault. What was that error? Explain why you think it actually caused the segmentation fault (this gets into some operating system issues that you might not have seen yet, but take a stab at it).

   Note that this error doesn't always manifest itself the same way on other variants of Unix - but on Linux in particular, it's been a source of a lot of hard-to-find errors I've had to track down.

3. The C compiler on Linux doesn't do much of a job of catching and reporting warnings for code that isn't strictly invalid C syntax but which does not make sense. Other compilers often have some code analysis built in to help catch some of these common errors. Fortunately, even if we're stuck with *gcc*/*g++*, there is a

tool available on Linux and most other Unix variants (and also on Windows and other platforms) that will analyze source code files and find all the stuff you're not supposed to do. What is this program? (Note that there are a lot of different variations on this program too, all with different names, but they usually have the same "fuzzy" root name.)

Run this program on each of the *err*.c* source files and see what it generates. Note that it will spit out plenty of other warnings too – try to sift out the one that was actually causing the error. Record the pertinent warning (or warnings) for each of the programs.

NOTE: you'll notice from the code that I'm pretty consistent in the way I format my source code, even for a little short throw-away program. If you're going to be writing code for a living, it's important to do a professional job of it. And that means not just making it run, but making it presentable. Trust me, in the future, when someone points you to a pile of source code that you have to debug and/or maintain, you'll really be hoping that the original author did the same. Earn some good karma and always write clean code.

I am going to apply this to your programming assignments – part of the points for each assignment will be based on how well-formatted and consistent your code is. That's not to say that I expect everyone to use the same coding style I do; a lot of programmer "holy wars" have been waged over the years about exactly what the "right" style is, and we're sure not going to answer that question. But you need to pick a style and apply it consistently.