Lab 2: How do I write this thing?

By now, you may be fighting a feeling of panic or impending doom. There is a way to do this lab in a time efficient manner that many students have used to get good results. A few guidelines may be in order:

- Design first, then code
- Don't try to do the whole thing at once
- Build a little, test a little
- Start early, work steadily (leave it and come back many times)

By Inclusion

The information in "Doing_lasbs_in_this_class.pdf" and "Thoughts_on_Lab_2_and_Single_Purpose" are officially part of this lab as well as lab 3 and lab 4. *Carefully* read the first of those and go over it.

Design

If you can't design it, you can't code it. Go over the write-up and start writing down questions. Go over it again and see if you can answer any of them and take the rest to office hours. Design, like code, follows the second bullet point above – don't do the whole thing at once. When designing at a higher level, presume the lower level stuff will do its job. You might want to start with the low-level stuff first and build up a set of "bricks" to make the final "wall." Most of us can't pick up a brick wall but nearly all of us can pick up one brick.

An example of low level design: Can you write a function that takes data for one object and prints it all on one line? This is the core brick for text output and once you've looked at printf, you can probably design and code this quite quickly. Once this handy brick is available, you can cut loose on the physics loop because you can print the object and see how the equations change it. You don't even have to get the formatting perfect the first time out of the box – that can be fixed later, but getting numbers out is critical to knowing what your software is doing.

Doing a small chunk, like creating an output routine, and then testing it supports the "build a little, test a little" paradigm. That particular brick of software further supports other build a little, test a little, efforts because it lets you see what happens to the numbers as you add to the physics code.

Kill Syntax Errors Early and Often

When writing C code for this lab, consider pausing after your write each function, saving the file, and compiling just that file. This is to get rid of syntax errors early and to get rid of them before they build up to a soul-crushing heap of errors. Consider this command: **make -r lab2.o** and think about what will happen. Kill syntax errors early, kill them often. The above command doesn't try to make the whole lab, it tries to make one object file. If you use multiple windows, this is easy. In the editor window, save the file but leave the editor live. For vi users that's **:w** and not **:~~wq~~**. In your other window, run make to build the .o file that corresponds to the C code file you are editing. Look at the syntax errors in one window and go fix them in the other window. We do it this way because right after you write a function, you are thinking about it. So compile it while it is already in your head. Compiling later means you have to go and figure out what it does all over again, wasting time. 4 windows open at a time is a really good idea in this class.