As with previous chapters, we are going to lay out the learning objectives for this chapter, and split them into what you should get on the first reading vs the second reading.

1 First Reading: Understand Key Ideas

On your first reading, the goal is to get the big ideas, and build the mental scaffolding in which to hold the details on your second reading—the learning objectives for your first reading are generally **Remember** and **Understand** objectives. If you are missing a few details (but not completely lost), it is ok to move ahead, and come back to them the next time through—you might ask questions on the course forums too! Note you should **sleep at least one night between the first and second reading** to give your brain time to process the information.

- **Compiling and Running: Intro**
  - **Define** *compile*.
  - **Explain** how to use `gcc` to compile your code.
- **The Compilation Process**
  - **List** the four major steps of compiling a C program.
  - **Explain** what the *preprocessor* does.
  - **Explain** the what the `#include` directive does and what is in header files.
  - **State** the difference between `#include <file.h>` and `#include "file.h"`.
  - **Explain** how to identify preprocessor directives in C source code.
  - **Define** *function prototype*.
  - **Explain** why function prototypes are useful.
  - **Define** *macro*
  - **Explain** what the `#define` directive does.
  - **Explain** three advantages of using macros for constants.
  - **Define** *portability*.
  - **Explain** the difference between calling a function and expanding a parameterized macro.
  - **Explain** pitfalls associated with macros.
  - **Define** *assembly*.
  - **Define** *parsing*.
  - **Explain** why the compiler may report later errors that are not actually problems.
  - **State** the order in which you should approach fixing compilation errors.
  - **Explain** why compilers may do a bad job describing the problem with erroneous code.
  - **Explain** what to do if the compiler's error messages use unfamiliar terms or otherwise don't make sense.
  - **Explain** how features of programmer's editors can help you avoid and/or quickly find and fix syntax mistakes.

  - **Define** *type checking*.
  - Note: not really a learning objective, but please note that Appendix F provides information about a lot of compiler error messages.
  - **Define** *optimize*.
  - **Define** *object file*.
  - **State** the compiler option to stop after compiling to an object file.
  - **Explain** why you might want compilation to stop after an object file.
  - **Define** *link*.
  - **Explain** how to recognize a linker error.
  - Note: you should encounter linker errors rarely enough that it will take you a while to fix them without consulting reference. You can certainly make notes on what was described here, but we'd recommend just coming back to this section if you have linker errors.
  - **State** what the `-l` option to gcc does.
  - **Explain** why short compilation times are important.
  - **Explain** what `make` does.

- **Running Your Program**
  - **Define** *PATH*.
  - **Explain** why you need to put ./ in front of your program's name to run it.
  - **Explain** why gdb is useful.
  - **Explain** why valgrind is useful.

- **Compiler Options**
  - **State** what the `-o` option to gcc does.
  - **Define** *compiler warning*.
  - **Explain** why we recommend `-Wall -Werror`.

- **Other Possibilities**
  - **Define** *interpret*.
  - **Explain** the difference between interpreting a program and compiling a program.
  - **Define** *REPL*.

2 Second Reading

This chapter is a bit different from most, in that it has a lot of factual information to help you understand what the compiler does and how to use it. We still recommend a second reading, as there is a lot of information to absorb.

Your main higher-level learning objective for this chapter (which you should *do* on the second reading when you get to Video 5.1: don't just watch the video, try to do this yourself) is the process of writing, compiling and running code, which involves these learning objectives:

- **Write code** in a programming editor (note: the specifics of using Emacs are covered in Appendix C).
- **Compile** your code with `gcc`.
- **Use** the `-o` and/or `-c` compiler options correctly when approrpiate.
- **Run** a program that you have compiled.

3 Longer Term Learning Objectives

You also should *start* working on these learning objectives. You won't master them during this chapter, but instead will need to work on them over the course of the semester, and probably beyond.

- **Select** approrpiate compiler options for a vareity of situations and features
- **Examine** compiler errors and determine what is wrong with your code.
- **Formulate** a plan to fix your code when it has compiler errors.

You will need to continuously work on these learning objectives as you write more code and encounter a variety of different errors. We want you to have the strategies (tips in the green boxes) for approaching errors in general, as well as the knowledge that we have put some reference about errors in Appendix F. Of course, you may encounter other errors that are not in Appendix F (or use a different compiler with different messages), so you should also remember to ask for help when you need it!