As with Chapter 1, 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.

- **Introduction**
  - **Explain** why reading code is an important skill
  - **List** the two important parts to reading code correctly:
  - **Define** *syntax*
  - **Define** *semantics*
  - **Define** *state of the program*
  - **Explain** what the "green arrow" means, and why it is typically between lines of code.
- **Variables**
  - **Define** *variable.*
  - **List** the two pieces of information needed to declare a variable.
  - **Define** *identifier.*
  - **State** the rules for valid identifiers in C.
  - **Define** *statement.*
  - **State** the syntax for declaring a variable in C.
  - **Explain** the semantics of declaring a variable in C.
  - **Define** *uninitialized.*
  - **Execute** C code by hand (by drawing a diagram, and moving the execution arrow) for code with one or more variable declarations.
  - **Define** *assignment statement.*
  - **Define** *lvalue.*
  - **State** the only kind of lvalue we have seen so far.
  - **Define** *expression*
  - **Explain** the difference between a *statement* and an *expression.*
  - **State** the syntax for an assignment statement in C.
  - **Execute** C code by hand for code with assignment statements.
  - **Define** *initialization.*

- **State** the syntax for initializing and declaring a variable in the same statement in C.
- **Execute** C code by hand for code with initialization and declaration of a variable combined into one statement.
- **Give examples** of valid C expressions using numeric constants, variables, +, -, *, /, and %, and parenthesis.
- **Define** *precedence*.
- **Define** *associativity*.
- **Define** *modulus operator*.
  Note: if you aren't familiar with modulus, don't get hung up on it. The learning objective to **compute** the value of expressions with modulus can wait until your second reading.
- **Explain** the differences between assignment statements and algebraic equalities.

- **Functions**
  - **Define** *function*.
  - **Define** *declaring a function*.
  - **Define** *calling a function*.
  - **Define** *return type* (of a function).
  - **Define** *parameter list* (of a function).
  - **Define** *body of a function*.
  - **State** the syntax of a function declaration.
  - **Explain** the similarities and differences between variables and parameters.
  - **Define** *return statement*
  - **State** the synatx of a return statement in C.
  - **Explain** what it means to use a function as an expression, and how that relates to the function's return value.
  - **Define** *stack frame*.
  - **Explain** what is special about the function named `main`.
  - **List** the steps for executing a function call and **explain** each one.
  - Note: we'll come back to the "execute code by hand" learning objectives for functions on the second reading
  - **Define** *scope*.
  - **Explain** why scope is useful to programmers.
  - **Define** *local variable*.
  - **Define** *block of code*.
  - **Explain** how to determine the scope of a variable.
  - **Recognize** that you should generally avoid global variables.
  - **Define** *string*.
  - **Explain** what the `printf` function does, including the `%d` specifier.

  - **Define** *escape sequences*.
  - **State** what the `\n` escape sequence means.

- **Conditional Statements**
  - **Define** *if/else statement*.
  - **Define** *conditional expression*.
  - **Explain** how true and false are represented in C.
  - **Compute** the value of expressions with the operators in Table 2.1.
  - **State** the syntax of an if/else statement.
  - **Define** *switch/case statement*.
  - **State** the syntax of a switch/case statement.
  - Note: we leave the details (break, fall-through, default) to the second reading.
  - **Recognize** that if/else and switch/case can be nested.

- **Shorthand**
  - **Define** *syntactic sugar*.
  - **Convert** shorthand forms to their meaning.
  - **Recognize** that leaving off curly braces around a block of code is risky.

- **Loops**
  - **Define** *loop*.
  - **State** the syntax for a while loop.
  - **State** the syntax for a do-while loop.
  - **State** the syntax for a for loop.
  - **Desugar** a for loop into an equivalent while loop.
  - **Recognize** that loops can be nested.
  - **Explain** what the `break` and `continue` statements do in a loop.

- **Higher Level Meaning**
  - **Recognize** that ascertaining higher-level meaning from code (figuring out what it does) is useful, but is a more advanced skill than covered here.

2 Second Reading: Be Able To Execute Code By Hand

As this chapter focuses on reading code, there are a lot of **Apply**-level learning objectives.

- **Compute** the value of expressions with modulus.
- **Execute** C code by hand for code with complex expressions on the righthand side of assignment statements.

- **Execute** C code by hand with function calls and returns
- **Show** the scope of a variable in a given piece of code.
- **Select** amongst variables of the same name, which one is referenced at a given point in the code based on the rules of scope.
- **Execute** C code by hand which has calls to `printf`
- **Execute** C code by hand which has if/else statements.
- **Explain** the concept of "fall through" in a switch statement
- **Explain** the concept of a "default case" in a switch statement
- **Execute** C code by hand which has switch/case statements.
- **Execute** C code by hand which has while loops, do-while loops, and/or for-loops.
- **Execute** C code by hand which has nested loops, or loops nested with if/else or with switch/case.
- **Execute** C code by hand which has break and/or continue in a loop.

3 Key Learning Objectives

The key learning object of this chapter is to **execute** C code by hand (drawing a diagram and updating it to reflect how state changes) for all the constructs we have seen in this chapter—variable declaration, assignment statements, evaluation of expressions, calling functions, returning values from a function, use of the return value as part of an expression, printing, if/else, conditional operators, switch/case (including break, fall through, and default), while loops, do-while loops, for loops, break in a loop, continue in a loop, and nesting constructs inside each other.

There are several practice questions at the end of the chapter that you can go through for more practice with these learning objectives.

Another learning objective that you should be *starting* on—but will take time and practice—is **finding** syntax errors in code and **evaluating** how to fix them.