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.

- **Strings**
  - Note: the first four of these were in Chapter 3. You should refresh the first 3 of them. The fourth (explain how strings are represented in C), you should now be able to do in more detail, as you have learned about arrays.
  - **Define** *string*.
  - **Define** *null terminator*.
  - **State** the syntax for the null terminator character literal in C.
  - **Explain** how strings are represented in C
  - **Explain** why strings are a subset of arrays of characters.
  - **State** the syntax for a string literal in C.
  - **State** the type of a string literal.
    - Note: you should be able to **explain** what that type means from prior Chapter 8.
  - **Explain** what happens if you leave the `const` qualifier off the type of a string literal.
  - **State** in which part of a program's address space string literals are stored.
  - Note: the details of the page table, and how regions of the address space can be made read only are outside the scope of this class.
  - **Explain** what happens if your program attempts to write into a read-only portion of its address space.
  - **Explain** *why* string literals are stored in a read-only portion of the address space.
  - **State** the syntax for declaring a mutable (modifiable) string literal in C.
  - **Explain** where the above syntax allocates space for the string, and why that string is mutable.
  - Note: there is a typo in AoP "The string literal on the left side is treated as an array..." should be "right" side.
  - **Explain** why we must be careful when providing an explicit size in the declaration of an array of characters to use as a string.
  - **Explain** why it is not good if something is actually broken, but the problem does not manifest in testing.

- **Explain** what `==` compares when its operands are strings.
- **Explain** what has to be done to compare two strings for equality.
- **Name** the C library function that compares strings for equality and ordering.
- **State** what `strcmp` returns and what each possible category of return values means.
- **Explain** what `=` does when its operands are strings.
- **Explain** what has to be done to copy a string.
- **Name** the C library function which copies strings up to a limited number of characters specified by one of its parameters.
- **Recognize** that care must be taken with `strncpy` to ensure the string is null-terminated if insufficient space might be available
  - If you read `man strncpy` on a Mac, it will recommend `strlcpy`, however, this function is not standard.
- **Recognize** that you should generally avoid `strcpy` (no `n`).
- **Explain** the difference between `"12345"` and `12345`.
- **Explain** some of the complexities in converting from a string to an integer.
- **Name** two C library functions which can convert a string to an int.
- **Explain** the difference between these to functions.
- **State** the general advice for when you should strongly suspect there is a C library function to do a particular task.

- **Multidimensional Arrays**
  - **Define** *multidimensional arrays.*
  - **Explain** why you might want to use multidimensional arrays to represent data.
  - **State** the syntax for declaring a multidimensional array.
  - **Explain** how a multidimensional array is laid out in memory.
  - **Explain** how to index a multidimensional array.
  - **State** the syntax for initializing a multidimensional array in its declaration.
  - Note: AoP describes some rules about when you can leave off one of the sizes. Don't get hung up on these rules—just always write down the sizes. AoP wants to be complete in case you ever see this in code.
  - **Explain** how multidimensional arrays generalize to more than 2 dimensions.
  - **Explain** how an array of pointers can be used to represent a multidimensional array.
  - **Explain** the differences between the two representations of multidimensional arrays.
  - Note: you should read "Incompatability of Representation" and watch Video 10.3 on both the first and second reading, but we will put all learning objectives for it on the second reading, as it is a bit more subtle and complex. Don't get stuck on it now.
  - **Explain** how arrays of strings work.
  - **Explain** the difference between arrays of strings represented as multidimensional arrays and represented as arrays of pointers.

  - **Explain** why it is useful and common to put NULL at the end of an array of strings represented as an array of pointers.

- **Function Pointers**
  - **Define** *function pointer*.
  - **Recognize** that a function's name is a pointer to that function.
  - **Explain** why function pointers are useful.
  - **State** the syntax to declare a variable (or parameter) whose type is a function pointer.
  - **State** the syntax to typedef a type which is another name for a function pointer type.
  - **Name** the C library function to sort an array.
  - **Explain** the parameters to `qsort`.

- **Security Hazards**
  - **Define** *security vulnerabilities*.
  - **Define** *buffer overflow*.
  - **Recognize** that `gets` is unsafe and you should NEVER use it.
  - **Define** *format string attacks*.
  - **Explain** how to avoid format string attacks.
  - **Define** *unsanitized inputs*.
  - **State** what backticks (`s) do in many command shells.
  - **Explain** SQL injection attacks.

2 Second Reading

As always, we strongly recommend you sleep between your first and second reading. The second reading is where you want to focus on the higher-level learning objectives, which build on the lower-level learning objectives you worked on in the first reading. Sleeping gives your brain a chance to process the information from the first reading.

- **Strings**
  - **Draw** a digram of the effects of declarations from code involving string literals, both as pointers, and as arrays in the frame (correctly based on the code).
  - **Execute** code by hand with strings, including `==` and `=` operators, and common C library functions.
  - **Show** how a textual representation of a number (any base) is converted to an integer.
  - **Develop** algorithms using strings
  - **Write** code using strings.
  - **Assess** when a C library function is appropriate for a desired string operation, and use that function correctly.

  - **Search** the man pages for appropriate C library functions.

- **Multidimensional Arrays**
  - **Draw** a diagram of the effects of declarations from code involving multidimensional arrays—either as true multidimensional arrays or as arrays of pointers—correctly based on the code.
  - **Execute** code by hand with multidimensional arrays.
  - **Select** the appropriate representation between multidimensional arrays and arrays of pointers.
  - **Develop** algorithms using multidimensional arrays.
  - **Write** code using multidimensional arrays (either "true" multidimensional arrays, or arrays of pointers).
  - **Explain** the differences in representation between true multidimensional arrays and arrays of pointers.
  - **Explain** why these two representations are incompatible.
  - **Execute** code by hand with arrays of strings.
  - **Develop** algorithms using arrays of strings.
  - **Write** code using arrays of strings.

- **Function Pointers**
  - **Execute** code with function pointers.
  - Note: over the longer term, you should **assess** when function pointers are appropriate to solving your problem, and **write** algorithms and code with function pointers. However, these are not a significant focus for right now, so don't get hung up on them.
  - Note: **drawing** diagrams of what happens for qsort's comparison function is great practice with pointers. Being able to **write** such functions for any datatype is a great longer term goal. However, those are not primary learning objectives for now.

- **Security Hazards**
  - We want you to have some general awareness of these security problems. However, this class does not have a focus on security. Depending on your career goals, we might strongly recommend a security class (*e.g.*, ECE 560) to go much deeper into these and many other topics.

3 Key Learning Objectives

The most key learning objectives for this chapter (which you won't master just by reading, but need to practice with) are:

- **Execute** code by hand that involves strings, multidimensional arrays, and arrays of pointers (including arrays of strings).

- **Develop** algorithms that involve strings, multidimensional arrays, and arrays of pointers (including arrays of strings).
- **Write** code that involve strings, multidimensional arrays, and arrays of pointers (including arrays of strings).