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.

- **Hardware Representation**
  - **Explain** the "everything is a number" principle and why it is important.
  - **Define** *bit*
  - **Explain** what a type specifies about the data it describes.
  - **Define** *binary number.*
  - **Explain** the value of each digit of a binary number.
  - **Explain** why computers use binary numbers.
  - **Define** *abstraction.*
  - **Explain** why abstraction is important.
  - **Define** *hexadecimal number.*
  - **Explain** why hexadecimal numbers are useful.
  - **Explain** how to interpret a hexidecimal number.
- **Basic Datatypes**
  - **Define** *compiler.*
  - **State** the number of bits in a *char.*
  - **Explain** what single quotation marks (as in `'A'`) mean.
  - **Explain** why you do NOT need to know the specific numeric values of characters in almost all situations.
  - **Define** *unsigned* (int).
  - **Define** *short* and *long* qualifiers on int.
  - **Define** *floating point number.*
  - **Define** *binary point.*
  - **Explain** how floating point numbers are represented.
  - **Explain** the difference between the *float* and *double* types.
  - **Explain** the pitfalls of comparing floating point numbers for equality.

- **Printing Redux**
  - **Define** *format specifier*
  - **Lookup** format specifiers in the **man pages** as needed.
    - Note: you will use `%d`, `%s`, and maybe `%c` fairly often, and soon be able to use them without looking things up. Please don't try to memorize all the format specifiers.
- **Expressions Have Types**
  - **Know** that expressions have types.
  - **State** the type of literal expressions.
  - **State** how to find the type of a variable used in an expression.
  - **Explain** how to find the type of an expression involving binary operators.
  - **State** how to find the type of a function call used in an expression.
  - **Explain** why the operands of a binary operator need to be converted to the same type if they are not already.
  - **Define** *type conversion.*
  - **Explain** what the compiler must do to perform a type conversion.
  - **Define** *sign extend.*
  - **Define** *zero extend.*
  - **Define** *truncate* (in the context of type conversions).
  - **Explain** which type conversion methods are appropriate under what circumstances.
  - **Define** *casting* (in the context of types).
  - **Explain** the difference between *type promotion* and *casting.*
  - **Explain** why you should be wary of casting.
  - **Explain** two circumstances under which you might need to cast.
  - **Explain** why dividing two integers and then assigning to a float gives the wrong result in many circumstances.
  - **Define** *overflow* and *underflow.*
  - **Explain** why it is incorrect to say that a *value* overflow or underflowed.
  - **Recognize** that in some cases overflow/underflow might be benign.
  - **Explain** why it is difficult and/or cumbersome to check for overflow everywhere.
  - **Explain** how programmers might guard against overflow.
- **"Non-Numbers"**
  - **List** some important kinds of data that do not naturally seem like numbers.
  - **Explain** why any type of data we want to use must be encoded as numbers.
  - **Define** *string*
  - **Define** *null terminator* and **state** how to write it down in C code.

  - **Explain** how a string is represented in C (note: this learning objective will be revisted in later chapters, for now a high-level explaination from the concepts in this chapter is sufficient).
  - **Define** *RGB*.
  - **Define** *pixel*.
  - **Explain** (at a high level) how images are encoded digitially.
  - **Explain** (at a high level) how sound is encoded digitially.
  - **Explain** (at a high level) how videos are encoded digitially.
  - **Explain** why compression is important for video encoding.

- **Complex, Custom Data Types**
  - **Define** *struct*
  - **State** the syntax rules at least one (your choice) of the ways to declare a struct.
  - **Explain** the dot (`.`) operator.
  - **Explain** what typedef does and why it is useful.
  - **State** the syntax of typedef.
  - **Define** *enumerated type*.
  - **State** the syntax for declaring and using enumerated types.
  - **Explain** why enumerated types are useful (hint: your answer should reference abstraction).

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.

- **Hardware Representation**
  - **Convert** positive binary numbers to decimal numbers.
  - **Convert** positive decimal numbers to binary numbers.
  - **Convert** hexadecimal numbers to binary numbers.
  - **Convert** binary numbers to hexadecimal numbers.
  - Note: converting decimal to/from hex is just a combination of the above skills.
  - **Separate** interface from implementation.
    - Note: you are starting on this learning objective here, but will practice/improve it over many years.

- **Basic Datatypes**

    - **Select** the appropriate built-in datatype to store data for algorithms you are writing.

- **Printing Redux**
    - **Evaluate** code which prints data with various format specifiers.
    - Note: we don't expect you to memorize the format specifiers. But you should quickly become familiar with the ones you use often, and you should be able to evaluate code with other specifiers given appropriate documentation (*e.g.*, the man pages) of what those specifiers do.

- **Expressions Have Types**
    - **Determine** the type of a (potentially complicated) C expression.
    - **Determine** when a C expression has a type error.
    - **Recommend** ways to change a C expression to fix a type error.
    - **Assess** when the types involved in an expression will cause it to evaluate to an undesired value and **recommend** how to change the expresion to evaluate properly.
    - Note: the above two learning objectives might involves recommending changes like storing data in different variable types, casting, storing intermediate results in a variable of an appropriate type, changing the return type of a function, or a variety of other strategies.
    - **Assess** if overflow/underflow are potential problems in a piece of code, and if so, **recommend** changes to the code to prevent the problem.

- **"Non-Numbers"**
    - **Construct** a representation of non-numeric data with numbers.
        * Note: we expect you to be starting on this LO now, not to totally master it. You should master it over the course of the semester.

- **Complex, Custom Data Types**
    - **Evaluate** C code with structs and the dot operator.
    - **Assess** when structs are appropriate to represent data.
    - **Construct** struct declarations to represent appropriate data, and **write** code to use those structs.
        * Note: At present, we expect you to just be able to do the basic mechanics of this LO. Converting algorithms to code is the subject of Chapter 4.
    - **Evaluate** C code with enums.
    - **Assess** when enums are appropriate to represent data.
    - **Construct** enum declarations to represent appropriate data, and **write** code to use those enums.
        * Note: At present, we expect you to just be able to do the basic mechanics of this LO. Converting algorithms to code is the subject of Chapter 4.

3 Key Learning Objectives

Here are the most key learning objectives from this chapter. Note that most of these will require practice (with upcoming assignments) and you won't just master them from reading:

- **Execute** code by hand which includes the new ideas in this chapter (types beyond int, printing in various formats, structs, dot operator, enums).
- **Determine** the type of an expression (or that there is a type error in it), **assess** when types cause unintended results, and **recommend** ways to fix type-related problems with your code.
- **Choose** the right datatype for your programming problem (which includes **assessing** concerns about overflow/underflow, the appropriateness of built-in types, the appropriateness of structs, the appropriateness of enums, and good abstraction).
- **Construct** the appropriate type to represent data whenever a built-in type is not appropriate.
- **Write** code which uses the various types you learned about in this chapter.
  - Note: At present, we expect you to just be able to do the basic mechanics of this LO. Converting algorithms to code is the subject of Chapter 4.