Top-down design

The process of taking large, complex pieces, and separating them out into their own function—known as ***top-down design*** —is crucial as you write larger and larger programs. Initially, we will write individual functions to serve a small, simple purpose—we might write one or two additional functions to implement a complex step. However, as your programming skill expands, you will write larger, more complex programs. Here, you may end up writing dozens of functions—solving progressively smaller problems until you reach a piece small enough that you do not need to break it down any further. While it may seem advantageous to just write everything in one giant function, such an approach not only makes the programming more difficult, but also tends to result in a complex mess that is difficult to test and debug. Whenever you have a chance to pull a well-defined logical piece of your program out into its own function, you should consider this an opportunity, not a burden. We will talk much more about writing larger programs later on after you master the basic programming concepts and are ready to write significant pieces of code.

Composability

When you are translating your code from your algorithmic description to C (or whatever other language you want), you can translate an instruction into code in the same way, no matter what other steps it is near, or what conditions or repetitions it is inside of. That is, you do not have to do anything special to write a loop inside of another loop, nor to write a conditional statement inside of a loop—you can just put the pieces together and they work as expected.

The ability to put things together and have them work as expected is called ***composability*** and is important to building not only programs, but other complex systems. If you put a for loop inside of an if statement, you do not need to worry about any special rules or odd behaviors: you only need to know how a for loop and an if statement work, and you can reason about the behavior of their combination.

In general, modern programming languages are designed so that features and language constructs can be composed, and work as expected. C (and later C++) follow this principle pretty well, so you can compose pretty much anything you learn from this book with pretty much anything else (with one notable exception that we will discuss later).