Analogy to Writing

So far, we have focused exclusively on programming in the small—designing the algorithm for a small-sized task (*i.e.*, one or a few functions), implementing it, testing it, and debugging it. However, most "real" programs have significant differences from the tiny ones we have developed so far. One key difference is that they tend to be much larger than those we have written (*e.g.*, the Linux kernel, has about 16 million lines of code in about 30 thousand source files). Another difference is that they have more than one person working on them, sometimes teams of hundreds to thousands. A third key difference is that real software has a long life-span during which it must be maintained.

Now that you have an understanding of the basics of programming in the small, we are ready to begin learning about programming in the large—the aspects of designing and implementing programs, which arise from having significantly-sized programs with long life-spans. A key aspect of programming in the large is the higher-level design of the program. Instead of just writing one (or a few) functions to perform a function-sized task, the programmer must now design the code into multiple appropriately-sized modules with clearly defined interfaces between them. Of course, programming in the small still comes into play, as the design will ultimately boil down to many function-sized tasks, which must be implemented.

A good analogy to think about programming in the small and in the large is writing (as in, English text). "Writing in the small" would be the task of crafting a sentence or paragraph. Here, you are concerned with issues like grammar (syntax) and word choice. You have one particular point you want to make, and you must make it clearly. This is about the size of the programming tasks you have accomplished so far.

Continuing our analogy, "writing in the large" would come into play for larger documents —whether they are dozen-page papers, or many-hundred-page books—which may also be written by multi-person teams. Now, we must be concerned with splitting the task into logical pieces (chapters and sections), and forming an outline. We must concern ourselves with the "interface" (in the case of writing: logical flow of ideas) between pieces at all granularities—chapter-to-chapter, section-to-section, and paragraph-to-paragraph. At some point, our "writing in the large" will produce a design that dictates many "writing in the small" tasks—we then apply our skill in that area to "implement" each of these paragraphs.

This analogy is particularly apt because the "in the large" and "in the small" skills are distinct, but go hand-in-hand in both cases. In both cases, you need both skills to write a complete document/program. You also need to keep the "in the small" in mind as you do the "in the large"—thinking about what makes for an appropriately-sized paragraph/function you can implement.

We are going to spend the rest of this module introducing programming in the large. Our coverage here will be sufficient for you to begin writing modest-sized programs, which we will start to work towards through the rest of this specialization. However, we will note that if you plan to work on very large software projects and/or want to become an expert in programming in the large, there is an entire sub-field of computer science, called Software Engineering, dedicated to this topic. In such a case, you should take classes in that area once you have mastered the basics of programming.

Completed