Translation to Code

Step 5: Translation to Code

Now that you are confident in your algorithm, it is time to translate it into code. This task is something that you can do with pencil and paper *e.g.*, as you often will need to do in a programming class on exams), but most of the time, you will want to actually type your code into an editor so that you can compile and run your program. Here, we will primarily focus on the mechanics of the translation from algorithmic steps to code. We *strongly* recommend that you acquaint yourself with a programmer's editor (Emacs or Vim) and use it whenever you program. We will cover Emacs in the next lesson, if you need an introduction.

Function declaration

We should start Step 5 by writing down the declaration of the function that we are writing, with its body (the code inside of it) replaced by the generalized algorithm from Step 3, written as *comments*. Comments are lines in a program which have a syntactic indication that they are for humans only (to make notes on how things work, and help people read and understand your code), and not an actual part of the behavior of the program. When you execute code by hand, you should simply skip over comments, as they have no effect. In C, there are two forms of comments: // comments to the end of the line, and /***...***/ makes everything between the slash-star and the star-slash into a comment.

One thing we may need to do in writing down the function declaration is to figure out its parameter types and return type. These may be given to us—in a class programming problem, you may be told as part of the assignment description and in a professional setting, it may be part of an interface agreed upon between members of the project team—however, if you do not know, you need to figure this out before proceeding.

Returning to our rectangle intersection example, we know that the function we are writing takes two rectangles, and returns a rectangle. Earlier, we decided that a rectangle could be represented as four numbers—suggesting a ***struct***. However, as you learned earlier, there are a variety of different types of numbers—should these numbers be ***int***s? ***float***s? ***double***s? Or some other type of number?

The answer to the question about which type of number we need is "It depends." You may be surprised to learn that "It depends" is often a perfectly valid answer to many questions related to programming, however, if you give this answer, you should describe what it depends on, and what the answer is under various circumstances.

For our rectangle example, the type that we need depends on what we are doing with the rectangles. One of the real number types ***float*** or ***double*** would make sense if we are writing a math-related program where our rectangles can have fractional coordinates. Choosing between ***float*** and ***double*** is a matter of what precision and what range we

need on our rectangles. If we are doing computer graphics, and working in the coordinates of the screen (which come in discrete pixels), then ***int*** makes the most sense, as you cannot have fractional pieces. For this example, we will assume that we want to use ***float***s.

With this decision made, we would start our translation to code by declaring the function and writing the algorithm in comments.

Completed
