Chapter 1
Why Programming and Why Perl?

This chapter provides two central premises for the rest of the book. First, why would a linguist, psycholinguist, literary theorist, and so on want to know anything about programming? Second, why would Perl be a good choice?

1.1 Why Programming?

Working with language data is nearly impossible these days without a computer. Data are massaged, analyzed, sorted, and distributed on computers. Various software packages are available for language researchers, but to truly take control of this domain, some amount of programming expertise is essential. Consider the following simple examples.

Imagine that you are a syntactician interested in the use of present-tense verbs. You have an electronic corpus and want to find all the cases of verbs in the present tense. How do you do it?

You're a literary stylist and want to investigate the distribution of words with iambic stress in Milton's poetry.

Imagine you are a phonologist. You're interested in consonant clusters. You have an electronic dictionary and want to find the largest word-final consonant cluster. Do you go through it by hand?

Finally, you're a psycholinguist and you want to perform an experiment investigating how people syllabify nonsense words.

All of these are fairly typical research tasks. If you don't know how to program yourself, you have only limited options. One possibility is to do the job by hand. For example, the syntactician could simply print out the corpus and go through it line by line. If the corpus is small enough, this might not be so onerous, but if the corpus is large, or if one really wants to be sure of one's

results, then this method is fraught with peril (and really boring). Another solution is to hire somebody else to do the job, but the same considerations apply. Yet a third possibility is to make use of some existing software package.

This last option is occasionally workable, but can fall short in several ways. First, an existing package is restricted by its design. That is, your needs may not match what the software was designed to do, rendering your task impossible or very difficult. Moreover, the software may not be intuitive, and may require learning some arcane set of commands or some difficult control language.[1] Finally, while software may exist to do what you want, it may not be available on the platform you work on (Windows, Mac, Unix), or may be too costly.

1.2 Why Perl?

The **Perl** programming language may provide an answer. There are a number of reasons why Perl may be an excellent choice.

First, Perl was designed for extracting information from text files. This makes it ideal for many of the kinds of tasks language researchers need.

Second, there are free Perl implementations for every type of computer. It doesn't matter what kind of operating system you use or computer architecture it's running on. There is a free Perl implementation available.

Third, it's *free*. Again, for any imaginable computer configuration, there is a free Perl implementation.

Fourth, it's extremely easy. In fact, it might not be an exaggeration to claim that of the languages that can do the kinds of things language researchers need, Perl may be the easiest to learn.

Fifth, Perl is an interpreted language. This means that you can write and run your programs immediately without going through an explicit intermediate stage to convert your program into something that the computer will understand.

Sixth, Perl is a natural choice for programming for the web. In chapter 9, I'll show how this presents some very useful opportunities to the language researcher.

Finally, Perl is a powerful programming language. While Perl is optimized for text manipulation, it can be used for just about anything else that one might want to do with a programming language.[2]

What this means is that learning all of Perl would be a monumental task. We won't let this deter us though. My strategy will be to pick and choose. I'll introduce those bits of Perl necessary to do the kinds of things people who work with language typically want to do. The rest – all the bells and whistles we don't need on our train – we'll leave for later. I'll let you know

where they are and how to find out more, but we won't digress to deal with them here.

1.3 Download and Install Perl

You may already have Perl on your system. If you're using some flavor of Unix, type `perl -v`. If you already have Perl, the program should display what version you have. It's possible that you have Perl, but that the program is not in your path. To check if it's anywhere on your system, you can use the `where` or `whereis` commands.

Under Windows, you should call up the **MS-DOS** prompt, and again type `perl -v`. If Perl is on your system, but not in your path, you can use the Windows `Find File` function to search for `perl.exe`.

For Macintosh, there is only one implementation of Perl, called **MacPerl**. Find the MacPerl icon and click on it.[3]

If you do not have Perl on your computer system, you can obtain it for free over the web. The following URL provides links to all implementations of Perl: `http://www.cpan.org`.

At the time of writing, the most recent version of Perl available is version 5. You should make sure that you have access to this version (or later), as the previous version (4) is lacking a number of important properties.

1.4 How to Read this Book

Learning to program isn't really hard, but you do need to do it the right way. The key is to start programming right away. As you read this book, you should make sure to try out the programs as we go through them. In fact, it would be ideal to read the book *at the computer*. Also, don't forget to try the exercises! You'll note that answers are not given at the end of the book. This is for two reasons. First, having answers is a big temptation. More importantly, however, most of the exercises involve revising or writing programs. There are often many ways to achieve the same goal and I would rather you find *some* way to answer an exercise question than feel you have to find *my* way of answering one of them.

Start by running the example programs exactly as given, either by downloading them from the website or, even better, by typing them in yourself. (Typing them in yourself will make the task familiar and draw your attention to aspects of the code you might miss otherwise.)

When you start to feel more comfortable, try varying the code a bit. The programs up through chapter 3 are perfectly safe and variations can't harm

your computer. After that point, certain operations should be handled with care, but I'll warn you about those as we go through.

The key, though, is to have fun!

Notes

[1] This latter point may seem analogous to learning a programming language, but notice that learning an arcane set of commands doesn't generalize; you would need to do that for every separate package that you have.

[2] The only place where Perl is lacking is in terms of graphics and graphical user interfaces. It's not possible to directly construct windows, buttons, and the like all in Perl. There are very reasonable ways around this limit, however. For example, as I discuss in appendix B, the optional `Tk` module allows for graphical user interfaces and other graphical programming.

[3] As of MacOS X, generic Unix Perl is available for Macs as well.