programming, you could look at the introductory section of the Mata manual or at the Mata chapters in Baum's friendly text *An Introduction to Stata Programming* (2016).

The examples in this book are statistical and mathematical. Formulas are provided, but the formulas are of secondary importance. They just provide the examples of something for us to program.

In this book, I will show you a language aimed at programming statistical and data management applications that has all the usual features and some unique ones, too. And I will show you programming techniques that might be new to you.

As I said, being serious is a matter of attitude. New techniques and languages are continually being developed, and you need to learn them, just as I still learn them. I have been programming for 45 years as a professional. I have a lot of experience and knowledge, but I have not stopped learning new techniques. I may be a professional programmer, but more importantly, I am a serious one.

1.2 What is Mata?

Many Stata users would describe Mata as a matrix language. StataCorp itself markets Mata that way. Mata would be more accurately described, however, as an across-platform portable-code compiled programming language that happens to have matrix capabilities. Just as important as its matrix capabilities are Mata's structures, classes, and pointers.

We at StataCorp designed and wrote Mata to be the development language that we would use. Nowadays, we write most new features of Stata in Mata. Before Mata existed, we used C. Compared with C, Mata code is easier to write, less error prone, easier to debug, and easier to maintain.

It is important that Mata is compiled. Being compiled means that programs run fast. Stata's other programming language, ado, is interpreted. Interpreted languages are slow in comparison with compiled languages. Mata code runs 10–40 times faster than ado.

Mata looks a lot like C and C++. In *The C Programming Language*, Kernighan and Ritchie (1978) introduced what has become perhaps the most famous first program:

```
main()
{
        printf("hello, world\n") ;
}
```

To convert the program to Mata, we need to add `void` in front of `main()`:

```
: void main()
> {
>         printf("hello, world\n") ;
> }
: main()
hello, world
```

Most Mata users would not bother typing the semicolon at the end of `printf("hello, world\n")`. Semicolons are optional in Mata. There are other differences between the languages, too. Those differences are covered in appendix C.

1.3 What is covered in this book

The programs we will write in this book are

Filename	Contents
`hello.mata`	First program, function `hello()`
`n_choose_k.mata`	Serious but short function, packaged as library function
`lr1.mata`	Linear regression, ver. 1 (structures)
`lr2.mata`	Linear regression, ver. 2 (structures)
`earthdistance.mata`	An aside concerning classes
`linreg1.mata`	Linear regression take 2, ver. 1 (classes)
`linreg2.mata`	Linear regression take 2, ver. 2 (classes)
`spmat1.mata`	Sparse matrices, ver. 1
`spmat2.mata`	Sparse matrices, ver. 2
`spmat3.mata`	Sparse matrices, ver. 3

The first serious program we will write is `n_choose_k()`. It will have just 47 lines including comments and white space.

We will then work our way to a nearly complete implementation of linear regression, starting with `lr1.mata` and ending with `linreg2.mata`. There will be only 388 lines in the final code in `linreg2.mata`! We will use structures for the first two implementations and use classes after that.

The `earthdistance.mata` program merely illustrates a point about class programming.

Finally, we will undertake a large project, namely, the implementation of sparse matrices. Sparse matrices are matrices in which most elements are 0. The project will concern storing the matrices efficiently—there is no reason to store all those 0s—and writing code to add and multiply them just as if they were regular matrices. File `spmat3.mata` will contain 937 lines.