Confessions of a Coding Monkey

Programming is usually taught by examples.
—Niklaus Wirth

Learning to program has one fatal flaw. It can be seriously boring. Sometimes it feels as though we have become mere *coding monkeys,* an endearing term used to describe programmers required to produce large amounts of code. In reality, programming is a skill that once learned, can be used to create innovative solutions to problems. Programming does not merely imply learning the syntax of a programming language and spending all your time coding. Many books spend a lot of time—whole chapters, in fact—describing programming constructs in detail. Many of these provide a good technical description of the constructs and syntax of the language but often fail to show how they can be used effectively. We choose to take a more simplistic approach that shows the syntax, followed by examples used to explain the concepts. At the end of the day, though, we can show you pages upon pages of examples, but programming is an art which must be practiced in order to develop a true understanding. The adage "practice makes perfect" holds true.

C is not an easy language with which to come to terms. Indeed, it was never designed as a teaching language. It has its idiosyncrasies and "features," which make you wonder what its designers Kernighan and Ritchie were thinking. Yet C has endured, and long after other languages have been laid to rest, C will still be around. There are many textbooks on C; indeed the bookshelf in my office is lined with them. Sometimes I wonder how so many books can all be so similar. This is not meant to be a traditional book on C, although we will go through the process of learning the traditional constructs associated with the language. This is more of a tutorial guide to the art of programming, for which we will use C as the instructional language. It could just as well be Java, Pascal, or MATLAB. The basic constructs of programming do not change. This is by no means an ordinary book on C programming. I always said I wouldn't write just another textbook on C. Everything should be reasonably easy to understand. This book is for the novice programmer, which is the place we all start. The only true experts on C are Kernighan, Ritchie, and those fortunate enough to have used C for the past 30 years. This book is meant to introduce the elements of programming *using* C as an illustrative language. It is by no means a definitive guide on C.

What sets this apart from other texts? Well, a good portion of time is spent looking at topics like programming style and debugging from the point of view of a CS1 student. The examples given tend to be nontraditional. The programs are usually all dissected, in that we look at each element of the program and sometimes explore alternatives. The case studies throughout the text are drawn from potential real-world applications.

A Note on Examples

The code examples given throughout the book have been compiled using Pelles C or Gnu Common C (gcc). They are not designed to be the most optimized solutions on the planet, nor are the programs cryptic. I don't use a lot of shortcuts in C. I find them terribly difficult for the novice to understand. Part of what I didn't want this book to become is a text on C programming syntax. Syntax is used throughout the book, but remember that it is there to illustrate a concept. An ideal book introducing programming would endeavor to discuss the ideas behind a concept such as loops and illustrate them. I hope that is achieved here. A mix of discussion of structures and concepts is used in programming with illustrations provided in C.

Nor did I want this to become a book interspersed with 1,001 irrelevant examples. What you will find are examples packed with algorithms from various fields of application. The idea is modeled somewhat on the early books of Niklaus Wirth, such as ***Pascal: User Manual and Report,*** which is coauthored with Kathleen Jensen and provides a nice balance between syntax and examples, all in 167 pages. If you want more on the syntax of C, I have included some excellent reference books. I don't hope to cover every syntactic structure in this book, for to do so would turn it into a reference manual. I have included sections on debugging strategies and even flowcharting. The former is rarely taught in an introductory course, and the latter is rarely practiced anymore—at least not in the style of programming. Surprisingly it rears its head in object-oriented programming, so why not start programmers off on the right footing?

This book is all about examples. Small snippets of code are used to illustrate programming constructs, larger programs to integrate ideas and show how a program is constructed. All of the examples are focused on a central notion of real-world applications. We talk about creating a simple encryption algorithm, which although insecure by today's standards, illustrates how programming can be used to advocate security. The chapters discussing concepts are offset by what we term *case studies*, which are chapters devoted to solving specific problems. The programs will become longer and a little more complex as the book progresses, but as your skills improve, a program that today looks daunting may look simpler.

Who Is This Book For?

This book is intended for the novice programmer, the person who never thought he or she would end up in a programming class. The thought of learning programming, let alone C may seem somewhat daunting, but this book provides a grounding in the art of programming in such a manner that you will be able to solve real-world problems by writing programs.

Commonly Asked Questions and Replies

I know C already.

Mention that the introductory language is C, and more often than not, what you'll hear is "Why not Java?" or "I already learned C in high school." Really? There are many levels of programmers. Some are novices, some are intermediate programmers, and a few are experts. Some programmers are proficient in a number of different languages, while others are masters of just one or two. In truth, the only true experts are those who design a language or spend 20 years programming in it on a daily basis. The rest of us may be proficient in a language, but may not be aware of all its intricacies. We use it to solve problems.

C is, like, soooo old.

Yes, C is old, if you count being over thirty as old. But it works. Let's face it, with introductory programming, you are learning more than just a particular language syntax. You are learning about how to design an algorithm and what structures to use. At the end of the day the particular programming language used is unimportant. It is just a way of illustrating the concepts. Yet most introductory courses do devolve into courses on language structures and syntax.

Why can't we learn a visual language, and when will we build GUIs?

Why, oh, why? Start with something simple. Rome wasn't built in a day, you know, and it lasted for more than a millennium! The same cannot be said about a lot of the software designed today. Besides, many programs that function inside systems like robots don't even have GUIs (graphical user interfaces).

Why don't we learn Java, or C#?

Why not teach Java or C#? Yes it would be tempting to teach one of these languages, but the crux is that Java is not a core language in industry and C# is, to quote Bjarne Stroustrup, "just another proprietary language." There are always new languages and most of them seem to be descendants of C in one form or another. This is not a book about languages per se, although you will learn to code in C. Our goal is to help you master the structures and nuances associated with programming.

Case Studies

The case studies are tutorials devoted to designing and implementing real-world algorithms. Each of the case studies builds on the material learned so far.

Case Study	Topics
1. Home Energy Footprint	program structure, equation, data types
2. Dam Forces	I/O, equations, data types, defensive
3. Estimating Dinosaur Speed and Gait	variables, equations, if-else
4. Humidex	I/O, statements, equations, variables
5. Power of a Wind Turbine	I/O, data types, equations, input validation
6. Radiometric Dating	functions, switch, if, while
7. Ariane 5	debugging
8. Heron's Formula	formula, equations, algorithm design
9. Leap Years	if-else, expressions
10. Population Ecology	I/O, equations
11. Fibonacci	algorithms, recursion, functions, arrays
12. Sieve of Eratosthenes	functions, loops, arrays
13. Roman Numerals	switch, loops, strings, arrays, functions
14. Caesar Cipher	functions, strings
15. Compound Interest	I/O, equations, loops
16. Identifying the Age of a Tree	searching, 2D strings, arrays
17. A Global Positioning System	functions, parameter passing
18. Parking Cars	recursion, random numbers
19. Image Processing	file I/O, 2D arrays, functions
20. Counting Words	characters, while, if-else
21. Great Circle Distance	equations, math functions
22. Bubble Sort	arrays, functions, nested loops
23. The Calendar Problem	arrays, algorithm refinement
24. Talking to Mars	equations, program structure
25. Matching DNA	functions, strings
26. Calculating π	loops, decisions, functions
27. Random Password Generator	random numbers, strings
28. Round-off Errors	mathematical errors
29. ISBN Validation	strings, usability, I/O, functions
30. Timber Reforestation	decisions, loops, testing

Note that not every case study is presented in the same way. Sometimes we derive the program piece-by-piece and then show the whole program, while other times we show the whole program and then dissect the program.