Preface

To the Student

I have learned the hard way that, when it comes to study habits, nothing is too obvious to state explicitly and repeatedly. Let me take this opportunity, at the start of a new voyage of discovery, to make a few suggestions.

First, reading passively is essentially useless. When reading this or any text, read with pencil in hand. Draw figures to help your understanding. **After reading through an example, close the text and try to reproduce the example.** If you cannot reproduce it, identify where you went wrong, study the text, and try again. Stop only when you can comfortably solve the example problem.

Second, incorporate lectures organically into the study process. Study the relevant reading before each lecture. Engage actively in lectures: take notes, ask questions, make observations. Laugh at the instructor's jokes. **The evening after each lecture, resolve the problems that were presented that day.** You will find that actively reviewing each lecture will solidify material beyond what you might now think is possible. Over the course of the semester, you will probably save time—and you will learn the material better than you would otherwise.

Third, solve exercises in the text even when they are not assigned. Use them to gauge your understanding of the material. If you are not confident that you solved a problem correctly, ask your peers for help or go to office hours. I have provided many exercises with solutions and explanations to facilitate an active approach to learning. Therefore, be active.

Finally, **address confusions immediately**. If you procrastinate on clearing up a point of confusion, it is likely to bite you again and again.

This book introduces a subject that is wide in scope. It focuses on concepts and techniques rather than listing how to use libraries and functions. Therefore, use Internet search engines to locate references on C libraries, particularly starting with Chapter 5; the `man` Unix utility to read about Unix programs; Internet search engines to learn how to use editors like `emacs` and

`vim`; the `help` command in `gdb`; and the `help` and `doc` commands in Matlab. Engineers must learn new powerful tools throughout their careers, so use this opportunity to learn *how to learn*.

To learn to program is to be initiated into an entirely new way of thinking about engineering, mathematics, and the world in general. Computation is integral to all modern engineering disciplines. The better you are at programming, the better you will be in your chosen field. Make the most of this opportunity. I promise that you will not regret the effort.

To the Instructor

This book departs radically from the typical presentation of programming: it presents pointers in the very first chapter—and thus in the first or second lecture of a course—as part of the development of a computational model. This model facilitates an ab initio presentation of otherwise mysterious subjects: function calls, call-by-reference, arrays, the stack, and the heap. Furthermore, it allows students to practice the essential skill of memory manipulation throughout the entire course rather than just at the end. Consequently, it is natural to go further in this text than is typical for a one-semester course: abstract data types and linked lists are covered in depth in Chapters 7 and 8. The computational model will also serve students in their adventures with programming beyond the course: instead of falling back on rules, they can think through the model to decide how a new programming concept fits with what they already know.

Another departure from the norm is the emphasis on programming from scratch. Most exercises do not provide starter code; the use of `gcc` and `make` are covered when appropriate. I expect students to leave the course knowing how to open a text editor, write one or multiple program files, compile the code, and execute and debug the resulting program. Many engineering students will not take an additional course on programming; hence, it is essential for them to know how to program from scratch after this course.

This book covers two programming languages: C and Matlab. The computational model and concepts of modularity are developed in the context of C. Matlab provides an engineering context in which students can transfer, and thus solidify, their mastery of programming from C. Matlab also provides an environment in which students, having learned how to create libraries in Chapters 6–8, can be critical users of libraries. They can think through how complex built-in functions and libraries might be implemented and thus learn techniques and patterns "on the job."

There are strong dependencies among chapters, except that Chapters 8 and 10 may be skipped. Furthermore, Chapter 4 is best left as a reading assignment. Of course, chapters may also be eliminated starting from the ending if time is in short supply.

Your results with my approach may vary. Certainly part of my success with this presentation of the material is a result of my aggressive teaching style and

the way that I organize my classes. Two studies in particular influence the way I approach teaching. The first investigates our ability, as students, to self-assess:

> Justin Kruger and David Dunning, *Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments*, J. of Personality and Social Psychology, v. 77, pp. 1121-1134, 1999.

The second addresses cause-and-effect in cheating and performance:

> David J. Palazzo, Young-Jin Lee, Rasil Warnakulasooriya, and David E. Pritchard, *Patterns, Correlates, and Reduction of Homework Copying*, Phys. Rev. ST Phys. Educ. Res., v. 6, n. 1, 2010.

My experience in the classroom having confirmed these studies, I administer hour-long quizzes every two to three weeks that test the material that students ought to have learned from the text, from lectures and labs, and from homework. Additionally, I give little weight to homework in the final grade. Therefore, students have essentially no incentive to cheat (themselves out of learning opportunities) on homework—and all the possible incentive to use homework to learn the material. Students have responded well to this structure. They appreciate the frequent feedback, and a significant subset attends office hours regularly. Fewer students fall behind. Consequently, I am able to fit all of the material in this book into one semester. In order to motivate students who start poorly, I announce mid-semester that the final exam grade can trump all quiz grades. Many students seem to learn what they need to know from the quizzes, and so many are better prepared for the final exam.

As side benefits, since enacting this teaching strategy in this and another course, I have never had to deal with an honor code violation—which is rare for introductory programming courses—and have not received a single complaint about a final grade, which is rarer still.

Acknowledgments

I developed the material for this book in preparation for and while teaching a first-year course on programming for engineering students at the University of Colorado, Boulder, partly with the support of an NSF CAREER award.[1] The course was offered in the Department of Electrical, Computer & Energy Engineering (ECEE) and also had students from the Department of Aerospace Engineering Sciences (AES). Thanks to Michael Lightner, the chair of ECEE, for allowing me to teach the course my way. I am grateful to the 77 students

[1] This material is based upon work supported by the National Science Foundation under grand No. 0952617. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

of the Spring 2011 offering for their patience with the new material—and for going along with the experiment and producing the best results of any class that I had taught up to that point. I also thank the teaching assistants—Arlen Cox, Justin Simmons, and Cary Goltermann—for their feedback on the material and on how the students were doing. Peter Mathys, a professor in ECEE, took the course and also provided excellent feedback.

Beyond the people already mentioned, thanks to those outside of the course who volunteered to read parts or all of the manuscript: Andrew Bradley, Caryn Sedloff, Sarah Solter, and Fabio Somenzi. Remaining errors, omissions, awkward phrasing, etc., are of course entirely my fault.

I am grateful to Zohar Manna, my PhD advisor and co-author of my first book, also published by Springer. Besides guiding my first foray into the world of crafting technical books, he showed me what work that stands the test of time looks like.

Sarah Solter, my wife and an accomplished professional software engineer, contributed in multiple ways. She acted as a sounding board for my ideas on how to present programming. As always, she supported me in my quest to do the right things well.

Finally, I thank Ronan Nugent and the other folks at Springer for once again being a supportive and friendly publisher.

ARB
Boulder, CO
June 2011

Contents

1 Memory: The Stack 1
- 1.1 Playing with Memory 2
  - 1.1.1 A First Foray into Programming 2
  - 1.1.2 Introduction to Pointers 4
  - 1.1.3 Pointers to Pointers 6
  - 1.1.4 How to Crash Your Program 11
- 1.2 Functions and the Stack 13
  - 1.2.1 Introduction to Functions 13
  - 1.2.2 A Protocol for Calling Functions 14
  - 1.2.3 Call-by-Value and Call-by-Reference 22
  - 1.2.4 Building Fences 25
- 1.3 Bits, Bytes, and Words 29

2 Control 31
- 2.1 Conditionals 31
- 2.2 Recursion 36
- 2.3 Loops 42

3 Arrays and Strings 47
- 3.1 Arrays 47
  - 3.1.1 Introduction to Arrays 47
  - 3.1.2 Looping over Arrays 50
  - 3.1.3 Arrays as Parameters 52
  - 3.1.4 Further Adventures with Arrays 54
- 3.2 Strings 61
  - 3.2.1 Strings: Arrays of `chars` 62
  - 3.2.2 Programming with Strings 63
  - 3.2.3 Further Adventures with Strings 67

4 Debugging 81
- 4.1 Write-Time Tricks and Tips 81
  - 4.1.1 Build Fences around Functions 81
  - 4.1.2 Document Code 83
  - 4.1.3 Prefer Readability to Cleverness 84
- 4.2 Compile-Time Tricks and Tips 84
- 4.3 Runtime Tricks and Tips 86
  - 4.3.1 GDB: The GNU Project Debugger 86
  - 4.3.2 Valgrind 92
- 4.4 A Final Word 92

5 I/O 93
- 5.1 Output 93
- 5.2 Input 97
  - 5.2.1 Command-Line Input 97
  - 5.2.2 Structured Input: Integer Data 101
  - 5.2.3 Structured Input: String Data 105
- 5.3 Working with Files 107
- 5.4 Further Adventures with I/O 107

6 Memory: The Heap 113
- 6.1 Review of Matrices 114
- 6.2 Matrix: A Specification 115
- 6.3 Matrix: An Implementation 120
  - 6.3.1 Defining the Data Structure 120
  - 6.3.2 Manipulating the Data Structure 128
- 6.4 Debugging Programs that Use the Heap 134

7 Abstract Data Types 137
- 7.1 Revisiting Matrices 138
- 7.2 FIFO Queue: A Specification 149
- 7.3 FIFO Queue: A First Implementation 154

8 Linked Lists 161
- 8.1 Introduction to Linked Lists 161
- 8.2 FIFO Queue: A Second Implementation 165
- 8.3 Priority Queue: A Specification 170
- 8.4 Priority Queue: An Implementation 173
- 8.5 Further Adventures with Linked Lists 178

9 Introduction to Matlab 181
- 9.1 The Command-Line Interface 182
- 9.2 Programming in Matlab 188
  - 9.2.1 Generating a Pure Tone 189
  - 9.2.2 Making Music 194

10 Exploring ODEs with Matlab 199
10.1 Developing an ODE Describing Orbits 199
10.1.1 Developing the ODE 199
10.1.2 Converting into a System of First-Order ODEs 201
10.2 Numerical Integration 202
10.3 Comparing Numerical Methods 205

11 Exploring Time and Frequency Domains with Matlab 215
11.1 Time and Frequency Domains 215
11.2 The Discrete Fourier Transform 219
11.3 De-hissing a Recording 228

Index 231