Praise for Higher-Order Perl . . .

As a programmer, your bookshelf is probably overflowing with books that did nothing to change the way you program . . . or think about programming.

You’re going to need a completely different shelf for this book.

While discussing caching techniques in Chapter 3, Mark Jason Dominus points out how a large enough increase in power can change the fundamental way you think about a technology. And that’s precisely what this entire book does for Perl.

It raids the deepest vaults and highest towers of Computer Science, and transforms the many arcane treasures it finds—recursion, iterators, filters, memoization, partitioning, numerical methods, higher-order functions, currying, curators, grammar-based parsing, lazy evaluation, and constraint programming—into powerful and practical tools for real-world programming tasks: file system interactions, HTML processing, database access, web spidering, typesetting, mail processing, home finance, text outlining, and diagram generation.

Along the way it also scatters smaller (but equally invaluable) gems, like the elegant explanation of the difference between “scope” and “duration” in Chapter 3, or the careful exploration of how best to return error flags in Chapter 4. It even has practical tips for Perl evangelists.

Dominus presents even the most complex ideas in simple, comprehensible ways, but never compromises on the precision and attention to detail for which he is so widely and justly admired.

His writing is—as always—lucid, eloquent, witty, and compelling.

Aply named, this truly is/a Perl book of a higher order, and essential reading for every serious Perl programmer.

—Damian Conway, Co-designer of Perl 6
HIGHER-ORDER PERL
TRANSFORMING PROGRAMS WITH PROGRAMS

Mark Jason Dominus
For Lorrie
CONTENTS

Preface ........................................................... xv

CHAPTER 1  Recursion and Callbacks ....................... 1
  1.1 DECIMAL TO BINARY CONVERSION ....................... 1
  1.2 FACTORIAL .................................................. 3
  1.2.1 Why Private Variables Are Important .................. 5
  1.3 THE TOWER OF HANOI ....................................... 6
  1.4 HIERARCHICAL DATA ......................................... 12
  1.5 APPLICATIONS AND VARIATIONS OF DIRECTORY WALKING .... 16
  1.6 FUNCTIONAL VERSUS OBJECT-ORIENTED PROGRAMMING .... 25
  1.7 HTML .......................................................... 26
  1.7.1 More Flexible Selection ................................. 32
  1.8 WHEN RECURSION BOWS UP .................................. 33
    1.8.1 Fibonacci Numbers ..................................... 33
    1.8.2 Partitioning ............................................. 35

CHAPTER 2  Dispatch Tables ..................................... 41
  2.1 CONFIGURATION FILE HANDLING ......................... 41
    2.1.1 Table-Driven Configuration ............................ 43
    2.1.2 Advantages of Dispatch Tables ......................... 45
    2.1.3 Dispatch Table Strategies ............................. 49
    2.1.4 Default Actions ......................................... 52
  2.2 CALCULATOR .................................................. 54
    2.2.1 HTML Processing Revisited ............................. 59

CHAPTER 3  Caching and Memoization ......................... 63
  3.1 CACHING FIXEDS RECURSION .............................. 65
  3.2 INLINE CACHING ............................................. 66
    3.2.1 Static Variables ......................................... 67
  3.3 GOOD IDEAS .................................................. 68
  3.4 MEMOIZATION ................................................ 69
4.3 EXAMPLES ................................................................. 126
  4.3.1 Permutations ...................................................... 128
  4.3.2 Genomic Sequence Generator .................................... 135
  4.3.3 Filehandle Iterators ................................................ 139
  4.3.4 A Flat-File Database ................................................ 140
  Improved Database ..................................................... 144
  4.3.5 Searching Databases Backwards .................................... 148
  A Query Package That Transforms Iterators ............................ 150
  An Iterator That Reads Files Backwards ................................ 152
  Putting It Together .................................................... 152
  4.3.6 Random Number Generation ...................................... 153
4.4 FILTERS AND TRANSFORMS ........................................ 157
  4.4.1 tmap() ......................................................... 158
  4.4.2 igrep() ..................................................... 160
  4.4.3 Tist_iterator() .............................................. 161
  4.4.4 append() ................................................... 162
4.5 THE SEMIPREDICATE PROBLEM .................................... 163
  4.5.1 Avoiding the Problem .......................................... 164
  4.5.2 Alternative undefs ............................................ 166
  4.5.3 Rewriting Utilities ............................................. 169
  4.5.4 Iterators That Return Multiple Values .......................... 170
  4.5.5 Explicit Exhaustion Function ................................... 171
  4.5.6 Four-Operation Iterators ..................................... 173
  4.5.7 Iterator Methods ............................................ 176
4.6 ALTERNATIVE INTERFACES TO ITERATORS ..................... 177
  4.6.1 Using foreach to Loop Over More Than One Array .......... 177
  4.6.2 An Iterator with an each-Like Interface ...................... 182
  4.6.3 Tied Variable Interfaces .................................... 184
     Summary of tie .................................................. 184
     Tied Scalars ................................................... 185
     Tied Filehandles ............................................... 186
4.7 AN EXTENDED EXAMPLE: WEB SPIDERS ............................. 187
  4.7.1 Pursuing Only Interesting Links ................................ 190
  4.7.2 Referring URLs .............................................. 192
  4.7.3 robots.txt .................................................. 197
  4.7.4 Summary .................................................... 200

CHAPTER 5 From Recursion to Iterators ......................... 203
5.1 THE PARTITION PROBLEM REVISITED ............................. 204
  5.1.1 Finding All Possible Partitions .............................. 206
## CONTENTS

5.1.2 Optimizations .............................................. 209
5.1.3 Variations ................................................. 212
5.2 HOW TO CONVERT A RECURSIVE FUNCTION TO AN ITERATOR ........................................... 215
5.3 A GENERIC SEARCH ITERATOR ................................................................. 225
5.4 OTHER GENERAL TECHNIQUES FOR ELIMINATING RECURSION ........................................... 229
5.4.1 Tail-Call Elimination ................................................................. 229
5.4.2 Creating Tail Calls ................................................................. 239
5.4.3 Explicit Stacks ................................................................. 242
5.4.4 Eliminating Recursion From \( \text{fib()} \) ................................................................. 243

### Chapter 6 Infinite Streams ................................................................. 255

6.1 LINKED LISTS ................................................................. 256
6.2 LAZY LINKED LISTS ................................................................. 257
6.2.1 A Trivial Stream: \( \text{upto()} \) ................................................................. 259
6.2.2 Utilities for Streams ................................................................. 260
6.3 RECURSIVE STREAMS ................................................................. 263
6.3.1 Memoizing Streams ................................................................. 265
6.4 THE HAMMING PROBLEM ................................................................. 269
6.5 REGEX STRING GENERATION ................................................................. 272
6.5.1 Generating Strings in Order ................................................................. 283
6.5.2 Regex Matching ................................................................. 286
6.5.3 Cutsorting ................................................................. 288
6.6 Log Files ................................................................. 293
6.6.1 Approximation Streams ................................................................. 300
6.6.2 Derivatives ................................................................. 305
6.6.3 The Tortoise and the Hare ................................................................. 308
6.6.4 Finance ................................................................. 310
6.7 POWER SERIES ................................................................. 313
6.7.1 Derivatives ................................................................. 319
6.7.2 Other Functions ................................................................. 320
6.7.3 Symbolic Computation ................................................................. 320

### Chapter 7 Higher-Order Functions and Currying ................................................................. 325

7.1 CURRYING ................................................................. 325
7.2 COMMON HIGHER-ORDER FUNCTIONS ................................................................. 333
7.2.1 Automatic Currying ................................................................. 335
7.2.2 Prototypes ................................................................. 337
7.3 Prototype Problems ................................................................. 338
7.2.3 More Currying .............................................. 340
7.2.4 Yet More Currying ........................................... 342
7.3 reduce() and combine() ........................................... 343
7.3.1 Boolean Operators ........................................... 348
7.4 DATABASES .................................................. 351
7.4.1 Operator Overloading ........................................... 356

CHAPTER 8 Parsing .................................................. 359
8.1 LEXERS .................................................................. 359
8.1.1 Emulating the <> Operator ..................................... 360
8.1.2 Lexers More Generally .......................................... 365
8.1.3 Chained Lexers ................................................. 368
8.1.4 Peeking .......................................................... 374
8.2 PARSING IN GENERAL .......................................... 376
8.2.1 Grammars ...................................................... 376
8.2.2 Parsing Grammars .............................................. 380
8.3 RECURSIVE-DESCENT PARSERS ................................ 384
8.3.1 Very Simple Parsers ........................................... 384
8.3.2 Parser Operators .............................................. 386
8.3.3 Compound Operators .......................................... 388
8.4 ARITHMETIC EXPRESSIONS .................................... 390
8.4.1 A Calculator .................................................. 400
8.4.2 Left Recursion ................................................ 400
8.4.3 A Variation on star() ......................................... 408
8.4.4 Generic-Operator Parsers ...................................... 412
8.4.5 Debugging ..................................................... 415
8.4.6 The Finished Calculator ....................................... 424
8.4.7 Error Diagnosis and Recovery ................................. 427
     Error-Recovery Parsers .......................................... 427
     Exceptions ........................................................ 430
8.4.8 Big Numbers .................................................. 435
8.5 PARSING REGEXES ............................................... 435
8.6 OUTLINES .......................................................... 440
8.7 DATABASE-QUERY PARSING ................................... 448
     8.7.1 The Lexer .................................................. 448
     8.7.2 The Parser ................................................ 451
8.8 BACKTRACKING PARSERS ...................................... 456
     8.8.1 Continuations ................................................ 457
     8.8.2 Parse Streams .............................................. 461
8.9 OVERLOADING .................................................. 465
CHAPTER 9  Declarative Programming ................................. 471
  9.1 constraint systems ........................................ 472
  9.2 Local Propagation Networks ................................ 472
      9.2.1 Implementing a Local Propagation Network ........ 475
      9.2.2 Problems with Local Propagation .................. 487
  9.3 linear equations ........................................... 488
  9.4 Hinogram: A Drawing System ............................... 490
      9.4.1 Equations ........................................... 500
          ref($base) || $base ................................... 501
          Solving Equations ..................................... 502
          Constraints ............................................. 512
      9.4.2 Values .............................................. 514
          Constant Values ....................................... 516
          Tuple Values .......................................... 518
          Feature Values ....................................... 520
          Intrinsic Constraints ................................ 521
          Synthetic Constraints ................................. 522
          Feature-Value Methods ................................. 527
      9.4.3 Feature Types ........................................ 530
          Scalar Types ........................................... 531
          Type Methods .......................................... 532
      9.4.4 The Parser .......................................... 539
          %TYPES ................................................. 541
          Programs .............................................. 543
          Definitions ............................................ 543
          Declarations ......................................... 545
          Expressions ............................................ 554
      9.4.5 Missing Features ..................................... 560
  9.5 conclusion .................................................. 563

Index ............................................................ 565

Function Index .................................................. 575
A well-known saying in the programming racket is that a good Fortran programer can write Fortran programs in any language. The sad truth, though, is that Fortran programmers write Fortran programs in any language whether they mean to or not. Similarly, we, as Perl programmers, have been writing C programs in Perl whether we meant to or not. This is a shame, because Perl is a much more expressive language than C. We could be doing a lot better, using Perl in ways undreamt of by C programmers, but we're not.

How did this happen? Perl was originally designed as a replacement for C on the one hand and Unix scripting languages like Bourne Shell and awk on the other. Perl's first major proponents were Unix system administrators, people familiar with C and with Unix scripting languages; they naturally tended to write Perl programs that resembled C and awk programs. Perl's inventor, Larry Wall, came from this sysadmin community, as did Randal Schwartz, his coauthor on Programming Perl, the first and still the most important Perl reference work. Other important early contributors include Tom Christiansen, also a C-and-Unix expert from way back. Even when Perl programmers didn't come from the Unix sysadmin community, they were trained by people who did, or by people who were trained by people who did.

Around 1993 I started reading books about Lisp, and I discovered something important: Perl is much more like Lisp than it is like C. If you pick up a good book about Lisp, there will be a section that describes Lisp's good features. For example, the book Paradigms of Artificial Intelligence Programming, by Peter Norvig, includes a section titled What Makes Lisp Different that describes seven features of Lisp. Perl shares six of these features; C shares none of them. These are big, important features, features like first-class functions, dynamic access to the symbol table, and automatic storage management. Lisp programmers have been using these features since 1957. They know a lot about how to use these language features in powerful ways. If Perl programmers can find out the things that Lisp programmers already know, they will learn a lot of things that will make their Perl programming jobs easier.

This is easier said than done. Hardly anyone wants to listen to Lisp programmers. Perl folks have a deep suspicion of Lisp, as demonstrated by Larry Wall's famous remark that Lisp has all the visual appeal of oatmeal with fingernail
clippings mixed in. Lisp programmers go around making funny noises like 'cons'
and 'cooder,' and they talk about things like the PC loser-ing problem, whatever
that is. They believe that Lisp is better than other programming languages, and
they say so, which is irritating. But now it is all okay, because now you do not
have to listen to the Lisp folks. You can listen to me instead. I will make sooth-
ing noises about hashes and stashes and globs, and talk about the familiar and
comforting soft reference and variable suicide problems. Instead of telling you
how wonderful Lisp is, I will tell you how wonderful Perl is, and at the end you
will not have to know any Lisp, but you will know a lot more about Perl.

Then you can stop writing C programs in Perl. I think that you will find it
to be a nice change. Perl is much better at being Perl than it is at being a slow
version of C. You will be surprised at what you can get done when you write Perl
programs instead of C.

WEB SITE

All the code examples in this book are available from my web site at:

http://perl.plover.com/hop/

When the notation in the margin is labeled with the tag some-example, the
code may be downloaded from:

http://perl.plover.com/hop/Examples/some-example

The web site will also carry the complete text, an errata listing, and other items
of interest. Should you wish to send me email about the book, please send your
message to mjd-hop@plover.com.

ACKNOWLEDGMENTS

Every acknowledgments section begins with a statement to the effect that "with-
out the untiring support and assistance from my editor, Tim Cox, this book
would certainly never have been written". Until you write a book, you will never
realize how true this is. Words fail me here; saying that the book would not
have been written without Tim's untiring support and assistance doesn't begin
to do justice to his contributions, his kindness, and his vast patience. Thank
you, Tim.
This book was a long time in coming, and Tim went through three assistants while I was working on it. All these people were helpful and competent, so my thanks to Brenda Modliszewski, Stacie Pierce, and Richard Camp. “Competent” may sound faint, but I consider it the highest praise.

Many thanks to Troy Lilly and Simon Crump, the production managers, who were not only competent but also fun to work with.

Shortly before the book went into production, I started writing tests for the example code. I realized with horror that hardly any of the programs worked properly. There were numerous small errors (and some not so small), inconsistencies between the code and the output, typos, and so on. Thanks to the heroic eleventh-hour efforts of Robert Spier, I think most of these errors have been caught. Robert was not only unfailingly competent, helpful, and productive, but also unfailingly cheerful, too. If any of the example programs in this book work as they should, you can thank Robert. (If they don’t, you should blame me, not Robert.) Robert was also responsible for naming the MOD document preparation system that I used to prepare the manuscript.

The contributions of my wife, Lorrie Kim, are too large and pervasive to note individually. It is to her that this book is dedicated.

A large number of other people contributed to this book, but many of them were not aware of it at the time. I was fortunate to have a series of excellent teachers, whose patience I must sometimes have tried terribly. Thanks to Mark Foster, Patrick X. Gallagher, Joan Livingston, Cal Lobel (who first taught me to program), Harry McLaughlin, David A. J. Meyer, Bruce Piper, Ronnie Rabassa, Michael Tempel, and Johan Tysk. Mark Foster also arrived from nowhere in the nick of time to suggest the title for this book just when I thought all was lost.

This book was directly inspired by two earlier books: *ML for the Working Programmer*, by Lawrence Paulson, and *Structure and Interpretation of Computer Programs*, by Harold Abelson and Gerald Jay Sussman. Other important influences were *Introduction to Functional Programming*, by Richard Bird and Philip Wadler, and *Paradigms of Artificial Intelligence Programming*, by Peter Norvig.

The official technical reviewers had a less rewarding job than they might have on other projects. This book took a long time to write, and although I wanted to have long conversations with the reviewers about every little thing, I was afraid that if I did that, I would never ever finish. So I rarely corresponded with the reviewers, and they probably thought that I was just filing their suggestions in the shredder. But I wasn’t: I pored over all their comments with the utmost care, and agonized over most of them. My thanks to the reviewers: Sean Burke, Damian Conway, Kevin Lenzo, Peter Norvig, Dan Schmidt, Kragen Sitaker, Michael Scott, and Adam Turoll.

While I was writing, I ran a mailing list for people who were interested in the book, and sent advance chapters to the mailing list. This was tremendously
helpful, and I'd recommend the practice to anyone else. The six hundred and fifty wonderful members of my mailing list are too numerous to list here. All were helpful and supportive, and the book is much better for their input. A few stand out as having contributed a particularly large amount of concrete material: Roland Young, Damien Warman, David “Novalis” Turner, Iain “Spoon” Truskett, Steve Tolkin, Ben Tilly, Rob Swirskas, Roses Longin Oduoung, Luc St-Louis, Jeff Mitchell, Steffen Müller, Abhijit Menon-Sen, Walt Mankowski, Wolfgang Laun, Paul Kulchenko, Daniel Koo, Andy Lester, David Landgren, Robin Houston, Torsten Hofmann, Douglas Hunter, Francesc Guasch, Kenneth Graves, Jeff Goff, Michael Fischer, Simon Cozens, David Combs, Stas Bekman, Greg Bacon, Darius Bacon, and Peter Allen. My apologies to the many many helpful contributors whom I have deliberately omitted from this list in the interests of space, and even more so to the several especially helpful contributors whom I have accidentally omitted.

Wolfgang Laun and Per Westerlund were particularly assiduous in helping me correct errors for the second printing.

Before I started writing, I received valuable advice about choosing a publisher from Philip Greenspun, Brian Kernighan, and Adam Turoff. Damian Conway and Abigail gave me helpful advice and criticism about my proposal.

Sean Burke recorded my Ivory Tower talk, cut CDs and sent them to me, and also supplied RTF-related consulting at the last minute. He also sent me periodic mail to remind me how wonderful my book was, which often arrived at times when I wasn’t so sure.

Several specific ideas in Chapter 4 were suggested by other people. Meng Wong suggested the clever and apt “odometer” metaphor. Randal Schwartz helped me with the “append” function. Eric Roode suggested the multiple list iterator.

When I needed to read out-of-print books by Paul Graham, A. E. Sundstrom lent them to me. When I needed a copy of volume 2 of The Art of Computer Programming, Hildo Biersma and Morgan Stanley bought it for me. When I needed money, B. B. King lent me some. Thanks to all of you.

The constraint system drawing program of Chapter 9 was a big project, and I was stuck on it for a long time. Without the timely assistance of Wm Leler, I might still be stuck.

Tom Christiansen, Jon Orwant, and Nat Torkington played essential and irreplaceable roles in integrating me into the Perl community.

Finally, the list of things “without which this book could not have been written” cannot be complete without thanking Larry Wall for writing Perl and for founding the Perl community, without which this book could not have been written.