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, cutsorting, 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.

Aptly 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

HIGHER-ORDER PERL

TRANSFORMING PROGRAMS WITH PROGRAMS

Mark Jason Dominus

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann Publishers is an imprint of Elsevier

MORGAN KAUFMANN PUBLISHERS

Senior Editor	Tim Cox
Publishing Services Manager	Simon Crump
Assistant Editor	Richard Camp
Cover Design	Yvo Riezebos Design
Cover Illustration	Yvo Riezebos Design
Composition	Cepha Imaging Pvt. Ltd.
Technical Illustration	Dartmouth Publishing, Inc.
Copyeditor	Eileen Kramer
Proofreader	Deborah Prato
Interior Printer	The Maple-Vail Book Manufacturing Group
Cover Printer	Phoenix Color

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data
Application submitted

ISBN: 1-55860-701-3

For information on all Morgan Kaufmann publications,
visit our Web site at www.mkp.com or www.books.elsevier.com

Printed in the United States of America
05 06 07 08 09 5 4 3 2 1

For Lorrie

CONTENTS

PREFACE

A well-known saying in the programming racket is that a good Fortran programmer 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 soothing 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 "without 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 Modliszewksi, 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 Turoff.

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 Svirskas, Roses Longin Odounga, 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.