SECOND EDITION

PROGRAMMING LANGUAGE PRAGMATICS

Michael L. Scott

Programming Language Pragmatics *is a very well-written textbook that captures the interest and focus of the reader. Each of the topics is very well introduced, developed, illustrated, and integrated with the preceding and following topics. The author employs up-to-date information and illustrates each concept by using examples from various programming languages. The level of presentation is appropriate for students, and the pedagogical features help make the chapters very easy to follow and refer back to*.

—Kamal Dahbur, *DePaul University*

Programming Language Pragmatics *strikes a good balance between depth and breadth in its coverage on of both classic and updated languages*.

—Jingke Li, *Portland State University*

Programming Language Pragmatics *is the most comprehensive book to date on the theory and implementation of programming languages. Prof. Scott writes well, conveying both unifying fundamental principles and the differing design choices found in today's major languages. Several improvements give this new second edition a more user-friendly format*.

—William Calhoun, *Bloomsburg University*

Prof. Scott has met his goal of improving Programming Language Pragmatics *by bringing the text up-to-date and making the material more accessible for students. The addition of the chapter on scripting languages and the use of XML to illustrate the use of scripting languages is unique in programming languages texts and is an important addition*.

—Eileen Head, *Binghamton University*

This new edition of Programming Language Pragmatics *does an excellent job of balancing the three critical qualities needed in a textbook: breadth, depth, and clarity. Prof. Scott manages to cover the full gamut of programming languages, from the oldest to the newest with sufficient depth to give students a good understanding of the important features of each, but without getting bogged down in arcane and idiosyncratic details. The new chapter on scripting languages is a most valuable addition as this class of languages continues to emerge as a major mainstream technology. This book is sure to become the gold standard of the field*.

—Christopher Vickery, *Queens College of CUNY*

Programming Language Pragmatics *not only explains language concepts and implementation details with admirable clarity, but also shows how computer architecture and compilers influence language design and implementation… This book shows that programming languages are the true center of computer science—the bridges spanning the chasm between programmer and machine*.

—*From the Foreword by* Jim Larus, *Microsoft Research*

Programming Language Pragmatics

SECOND EDITION

About the Author

Michael L. Scott is a professor and past chair of the Department of Computer Science at the University of Rochester. He received his Ph.D. in computer sciences in 1985 from the University of Wisconsin–Madison. His research interests lie at the intersection of programming languages, operating systems, and high-level computer architecture, with an emphasis on parallel and distributed computing. He is the designer of the Lynx distributed programming language and a codesigner of the Charlotte and Psyche parallel operating systems, the Bridge parallel file system, and the Cashmere and InterWeave shared memory systems. His MCS mutual exclusion lock, codesigned with John Mellor-Crummey, is used in a variety of commercial and academic systems. Several other algorithms, codesigned with Maged Michael and Bill Scherer, appear in the `java.util.concurrent` standard library.

Dr. Scott is a member of the Association for Computing Machinery, the Institute of Electrical and Electronics Engineers, the Union of Concerned Scientists, and Computer Professionals for Social Responsibility. He has served on a wide variety of program committees and grant review panels, and has been a principal or coinvestigator on grants from the NSF, ONR, DARPA, NASA, the Departments of Energy and Defense, the Ford Foundation, Digital Equipment Corporation (now HP), Sun Microsystems, Intel, and IBM. He has contributed to the GRE advanced exam in computer science, and is the author of some 95 refereed publications. In 2003 he chaired the ACM Symposium on Operating Systems Principles. He received a Bell Labs Doctoral Scholarship in 1983 and an IBM Faculty Development Award in 1986. In 2001 he received the University of Rochester's Robert and Pamela Goergen Award for Distinguished Achievement and Artistry in Undergraduate Teaching.

Programming Language Pragmatics

SECOND EDITION

Michael L. Scott

Department of Computer Science
University of Rochester

ELSEVIER

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

Publishing Director: Michael Forster
Publisher: Denise Penrose
Publishing Services Manager: Andre Cuello
Assistant Publishing Services Manager
Project Manager: Carl M. Soares
Developmental Editor: Nate McFadden
Editorial Assistant: Valerie Witte
Cover Design: Ross Carron Designs
Cover Image: © Brand X Pictures/Corbin Images
Text Design: Julio Esperas
Composition: VTEX
Technical Illustration: Dartmouth Publishing Inc.
Copyeditor: Debbie Prato
Proofreader: Phyllis Coyne et al. Proofreading Service
Indexer: Ferreira Indexing Inc.
Interior printer: Maple-Vail
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 13: 978-0-12-633951-2
ISBN10: 0-12-633951-1

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

To the roses now in full bloom.

Foreword

Computer science excels at layering abstraction on abstraction. Our field's facility for hiding details behind a simplified interface is both a virtue and a necessity. Operating systems, databases, and compilers are very complex programs shaped by forty years of theory and development. For the most part, programmers need little or no understanding of the internal logic or structure of a piece of software to use it productively. Most of the time, ignorance is bliss.

Opaque abstraction, however, can become a brick wall, preventing forward progress, instead of a sound foundation for new artifacts. Consider the subject of this book, programs and programming languages. What happens when a program runs too slowly, and profiling cannot identify any obvious bottleneck or the bottleneck does not have an algorithmic explanation? Some potential problems are the translation of language constructs into machine instructions or how the generated code interacts with a processor's architecture. Correcting these problems requires an understanding that bridges levels of abstraction.

Abstraction can also stand in the path of learning. Simple questions—how programs written in a small, stilted subset of English can control machines that speak binary or why programming languages, despite their ever growing variety and quantity, all seem fairly similar—cannot be answered except by diving into the details and understanding computers, compilers, and languages.

A computer science education, taken as a whole, can answer these questions. Most undergraduate programs offer courses about computer architecture, operating systems, programming language design, and compilers. These are all fascinating courses that are well worth taking—but difficult to fit into most study plans along with the many other rich offerings of an undergraduate computer science curriculum. Moreover, courses are often taught as self-contained subjects and do not explain a subject's connections to other disciplines.

This book also answers these questions, by looking beyond the abstractions that divide these subjects. Michael Scott is a talented researcher who has made major contributions in language implementation, run-time systems, and computer architecture. He is exceptionally well qualified to draw on all of these fields

to provide a coherent understanding of modern programming languages. This book not only explains language concepts and implementation details with admirable clarity, but also shows how computer architecture and compilers influence language design and implementation. Moreover, it neatly illustrates how different languages are actually used, with realistic examples to clearly show how problem domains shape languages as well.

In interest of full disclosure, I must confess this book worried me when I first read it. At the time, I thought Michael's approach de-emphasized programming languages and compilers in the curriculum and would leave students with a superficial understanding of the field. But now, having reread the book, I have come to realize that in fact the opposite is true. By presenting them in their proper context, this book shows that programming languages are the true center of computer science—the bridges spanning the chasm between programmer and machine.

James Larus, Microsoft Research

Contents

Preface

A course in computer programming provides the typical student's first exposure to the field of computer science. Most students in such a course will have used computers all their lives, for e-mail, games, web browsing, word processing, instant messaging, and a host of other tasks, but it is not until they write their first programs that they begin to appreciate how applications work. After gaining a certain level of facility as programmers (presumably with the help of a good course in data structures and algorithms), the natural next step is to wonder how programming languages work. This book provides an explanation. It aims, quite simply, to be the most comprehensive and accurate languages text available, in a style that is engaging and accessible to the typical undergraduate. This aim reflects my conviction that students will understand more, and enjoy the material more, if we explain what is really going on.

In the conventional "systems" curriculum, the material beyond data structures (and possibly computer organization) tends to be compartmentalized into a host of separate subjects, including programming languages, compiler construction, computer architecture, operating systems, networks, parallel and distributed computing, database management systems, and possibly software engineering, object-oriented design, graphics, or user interface systems. One problem with this compartmentalization is that the list of subjects keeps growing, but the number of semesters in a bachelor's program does not. More important, perhaps, many of the most interesting discoveries in computer science occur at the boundaries *between* subjects. The RISC revolution, for example, forged an alliance between computer architecture and compiler construction that has endured for 20 years. More recently, renewed interest in virtual machines has blurred the boundary between the operating system kernel and the language run-time system. The spread of Java and .NET has similarly blurred the boundary between the compiler and the run-time system. Programs are now routinely embedded in web pages, spreadsheets, and user interfaces.

Increasingly, both educators and practitioners are recognizing the need to emphasize these sorts of interactions. Within higher education in particular there is

a growing trend toward integration in the core curriculum. Rather than give the typical student an in-depth look at two or three narrow subjects, leaving holes in all the others, many schools have revised the programming languages and operating systems courses to cover a wider range of topics, with follow-on electives in various specializations. This trend is very much in keeping with the findings of the ACM/IEEE-CS *Computing Curricula 2001* task force, which emphasize the growth of the field, the increasing need for breadth, the importance of flexibility in curricular design, and the overriding goal of graduating students who "have a system-level perspective, appreciate the interplay between theory and practice, are familiar with common themes, and can adapt over time as the field evolves" [CR01, Sec. 11.1, adapted].

The first edition of *Programming Language Pragmatics* (PLP-1e) had the good fortune of riding this curricular trend. The second edition continues and strengthens the emphasis on integrated learning while retaining a central focus on programming language design.

At its core, PLP is a book about *how programming languages work*. Rather than enumerate the details of many different languages, it focuses on concepts that underlie all the languages the student is likely to encounter, illustrating those concepts with a variety of concrete examples, and exploring the tradeoffs that explain *why* different languages were designed in different ways. Similarly, rather than explain how to build a compiler or interpreter (a task few programmers will undertake in its entirety), PLP focuses on what a compiler does to an input program, and why. Language design and implementation are thus explored together, with an emphasis on the ways in which they interact.

Changes in the Second Edition

There were four main goals for the second edition:

1. Introduce new material, most notably scripting languages.
2. Bring the book up to date with respect to everything else that has happened in the last six years.
3. Resist the pressure toward rising textbook prices.
4. Strengthen the book from a pedagogical point of view, to make it more useful and accessible.

Item (1) is the most significant change in content. With the explosion of the World Wide Web, languages like Perl, PHP, Tcl/Tk, Python, Ruby, JavaScript, and XSLT have seen an enormous upsurge not only in commercial significance, but also in design innovation. Many of today's graduates will spend more of their time working with scripting languages than with C++, Java, or C#. The new chapter on scripting languages (Chapter 13) is organized first by application domain (shell languages, text processing and report generation, mathematics and statistics, "glue" languages and general purpose scripting, extension languages, script-

ing the World Wide Web) and then by innovative features (names and scopes, string and pattern manipulation, high level data types, object orientation). References to scripting languages have also been added wherever appropriate throughout the rest of the text.

Item (2) reflects such key developments as the finalized C99 standard and the appearance of Java 5 and C# (version 2.0). Chapter 6 (Control Flow) now covers boxing, unboxing, and the latest iterator constructs. Chapter 8 (Subroutines) covers Java and C# generics. Chapter 12 (Concurrency) covers the Java 5 concurrency library (JSR 166). References to C# have been added where appropriate throughout. In keeping with changes in the microprocessor market, the ubiquitous Intel/AMD x86 has replaced the Motorola 68000 in the case studies of Chapters 5 (Architecture) and 8 (Subroutines). The MIPS case study in Chapter 8 has been updated to 64-bit mode. References to technological constants and trends have also been updated. In several places I have rewritten examples to use languages with which students are more likely to be familiar; this process will undoubtedly continue in future editions.

Many sections have been heavily rewritten to make them clearer or more accurate. These include coverage of finite automaton creation (2.2.1); declaration order (3.3.3); modules (3.3.4); aliases and overloading (3.6.1 and 3.6.2); polymorphism and generics (3.6.3, 7.1.2, 8.4, and 9.4.4); separate compilation (3.7); continuations, exceptions, and multilevel returns (6.2.1, 6.2.2, and 8.5); calling sequences (8.2); and most of Chapter 5.

Item (3) reflects Morgan Kaufmann's commitment to making definitive texts available at student-friendly prices. PLP-1e was larger and more comprehensive than competing texts, but sold for less. This second edition keeps a handle on price (and also reduces bulk) with high-quality paperback construction.

Finally, item (4) encompasses a large number of presentational changes. Some of these are relatively small. There are more frequent section headings, for example, and more historical anecdotes. More significantly, the book has been organized into four major parts:

Part I covers foundational material: (1) Introduction to Language Design and Implementation; (2) Programming Language Syntax; (3) Names, Scopes, and Bindings; (4) Semantic Analysis; and (5) Target Machine Architecture. The second and fifth of these have a fairly heavy focus on implementation issues. The first and fourth are mixed. The third introduces core issues in language design.

Part II continues the coverage of core issues: (6) Control Flow; (7) Data Types; (8) Subroutines and Control Abstraction; and (9) Data Abstraction and Object Orientation. The last of these has moved forward from its position in PLP-1e, reflecting the centrality of object-oriented programming to much of modern computing.

Part III turns to alternative programming models: (10) Functional Languages; (11) Logic Languages; (12) Concurrency; and (13) Scripting Languages. Functional and logic languages shared a single chapter in PLP-1e.

Part IV returns to language implementation: (14) Building a Runnable Program (code generation, assembly, and linking); and (15) Code Improvement (optimization).

The PLP CD

To minimize the physical size of the text, make way for new material, and allow students to focus on the fundamentals when browsing, approximately 250 pages of more advanced or peripheral material has been moved to a companion CD. For the most part (though not exclusively), this material comprises the sections that were identified as advanced or optional in PLP-1e.

The most significant single move is the entire chapter on code improvement (15). The rest of the moved material consists of scattered, shorter sections. Each such section is represented in the text by a brief introduction to the subject and an "In More Depth" paragraph that summarizes the elided material.

Note that the placement of material on the CD does *not* constitute a judgment about its technical importance. It simply reflects the fact that there is more material worth covering than will fit in a single volume or a single course. My intent is to retain in the printed text the material that is likely to be covered in the largest number of courses.

Design & Implementation Sidebars

PLP-1e placed a heavy emphasis on the ways in which language design constrains implementation options, and the ways in which anticipated implementations have influenced language design. PLP-2e uses more than 120 sidebars to make these connections more explicit. A more detailed introduction to these sidebars appears on page 7 (Chapter 1). A numbered list appears in Appendix B.

Numbered and Titled Examples

Examples in PLP-2e are intimately woven into the flow of the presentation. To make it easier to find specific examples, to remember their content, and to refer to them in other contexts, a number and a title for each is now displayed in a marginal note. There are nearly 900 such examples across the main text and the CD. A detailed list appears in Appendix C.

Exercise Plan

PLP-1e contained a total of 385 review questions and 312 exercises, located at the ends of chapters. Review questions in the second edition have been moved to the

ends of sections, closer to the material they cover, to make it easier to tell when one has grasped the central concepts. The total number of such questions has nearly doubled.

The problems remaining at the ends of chapters have now been divided into *Exercises* and *Explorations*. The former are intended to be more or less straightforward, though more challenging than the per-section review questions; they should be suitable for homework or brief projects. The exploration questions are more open-ended, requiring web or library research, substantial time commitment, or the development of subjective opinion. The total number of questions has increased from a little over 300 in PLP-1e to over 500 in the current edition. Solutions to the exercises (but not the explorations) are available to registered instructors from a password-protected web site: visit *www.mkp.com/companions/0126339511/*.

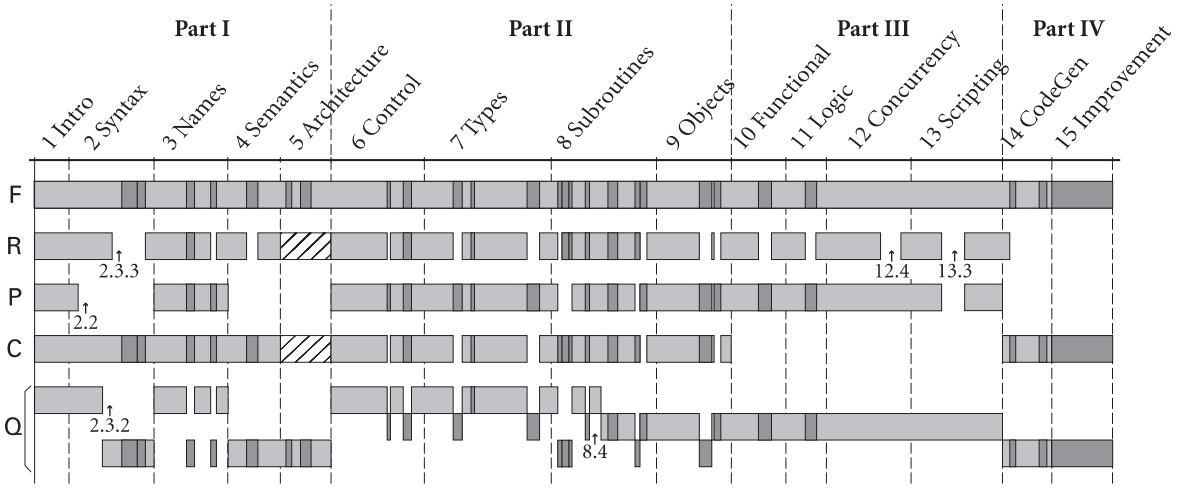
How to Use the Book

Programming Language Pragmatics covers almost all of the material in the PL "knowledge units" of the *Computing Curricula 2001* report [CR01]. The book is an ideal fit for the *CS 341* model course (Programming Language Design), and can also be used for *CS 340* (Compiler Construction) or *CS 343* (Programming Paradigms). It contains a significant fraction of the content of *CS 344* (Functional Programming) and *CS 346* (Scripting Languages). Figure 1 illustrates several possible paths through the text.

For self-study, or for a full-year course (track F in Figure 1), I recommend working through the book from start to finish, turning to the PLP CD as each "In More Depth" section is encountered. The one-semester course at the University of Rochester (track R), for which the text was originally developed, also covers most of the book but leaves out most of the CD sections, as well as bottom-up parsing (2.3.3), message passing (12.4), web scripting (13.3), and most of Chapter 14 (Building a Runnable Program).

Some chapters (2, 4, 5, 14, 15) have a heavier emphasis than others on implementation issues. These can be reordered to a certain extent with respect to the more design-oriented chapters, but it is important that Chapter 5 or its equivalent be covered before Chapters 6 through 9. Many students will already be familiar with some of the material in Chapter 5, most likely from a course on computer organization. In this case the chapter may simply be skimmed for review. Some students may also be familiar with some of the material in Chapter 2, perhaps from a course on automata theory. Much of this chapter can then be read quickly as well, pausing perhaps to dwell on such practical issues as recovery from syntax errors, or the ways in which a scanner differs from a classical finite automaton.

A traditional programming languages course (track P in Figure 1) might leave out all of scanning and parsing, plus all of Chapters 4 and 5. It would also deemphasize the more implementation-oriented material throughout. In place



F: The full-year/self-study plan
R: The one-semester Rochester plan
P: The traditional Programming Languages plan; would also de-emphasize implementation material throughout the chapters shown
C: The compiler plan; would also de-emphasize design material throughout the chapters shown
Q: The 1+2 quarter plan: an overview quarter and two independent, optional follow-on quarters, one language-oriented, the other compiler-oriented

Supplemental (CD) section
To be skimmed by students in need of review

Figure 1 Paths through the text. Darker shaded regions indicate supplemental "In More Depth" sections on the PLP CD. Section numbers are shown for breaks that do not correspond to supplemental material.

of these it could add such design-oriented CD sections as the ML type system (7.2.4), multiple inheritance (9.5), Smalltalk (9.6.1), lambda calculus (10.6), and predicate calculus (11.3).

PLP has also been used at some schools for an introductory compiler course (track C in Figure 1). The typical syllabus leaves out most of Part III (Chapters 10 through 13), and deemphasizes the more design-oriented material throughout. In place of these it includes all of scanning and parsing, Chapters 14 and 15, and a slightly different mix of other CD sections.

For a school on the quarter system, an appealing option is to offer an introductory one-quarter course and two optional follow-on courses (track Q in Figure 1). The introductory quarter might cover the main (non-CD) sections of Chapters 1, 3, 6, and 7, plus the first halves of Chapters 2 and 8. A language-oriented follow-on quarter might cover the rest of Chapter 8, all of Part III, CD sections from Chapters 6 through 8, and possibly supplemental material on formal semantics, type systems, or other related topics. A compiler-oriented follow-on quarter might cover the rest of Chapter 2; Chapters 4–5 and 14–15, CD sec-

tions from Chapters 3 and 8–9, and possibly supplemental material on automatic code generation, aggressive code improvement, programming tools, and so on.

Whatever the path through the text, I assume that the typical reader has already acquired significant experience with at least one imperative language. Exactly which language it is shouldn't matter. Examples are drawn from a wide variety of languages, but always with enough comments and other discussion that readers without prior experience should be able to understand easily. Single-paragraph introductions to some 50 different languages appear in Appendix A. Algorithms, when needed, are presented in an informal pseudocode that should be self-explanatory. Real programming language code is set in `"typewriter" font`. Pseudocode is set in a sans-serif font.

Supplemental Materials

In addition to supplemental sections of the text, the PLP CD contains a variety of other resources:

- Links to language reference manuals and tutorials on the Web
- Links to Open Source compilers and interpreters
- Complete source code for all nontrivial examples in the book (more than 300 source files)
- Search engine for both the main text and the CD-only content

Additional resources are available at *www.mkp.com/companions/0126339511/* (you may wish to check back from time to time). For instructors who have adopted the text, a password-protected page provides access to

- Editable PDF source for all the figures in the book
- Editable PowerPoint slides
- Solutions to most of the exercises
- Suggestions for larger projects

Acknowledgments for the Second Edition

In preparing the second edition I have been blessed with the generous assistance of a very large number of people. Many provided errata or other feedback on the first edition, among them Manuel E. Bermudez, John Boyland, Brian Cumming, Stephen A. Edward, Michael J. Eulenstein, Tayssir John Gabbour, Tommaso Galleri, Eileen Head, David Hoffman, Paul Ilardi, Lucian Ilie, Rahul Jain, Eric Joanis, Alan Kaplan, Les Lander, Jim Larus, Hui Li, Jingke Li, Evangelos Milios, Eduardo Pinheiro, Barbara Ryder, Nick Stuifbergen, Raymond Toal, Andrew Tolmach, Jens Troeger, and Robbert van Renesse. Zongyan Qiu prepared the Chinese translation, and found several bugs in the process. Simon Fillat maintained

the Morgan Kaufmann web site. I also remain indebted to the many other people, acknowledged in the first edition, who helped in that earlier endeavor, and to the reviewers, adopters, and readers who made it a success. Their contributions continue to be reflected in the current edition.

Work on the second edition began in earnest with a "focus group" at SIGCSE'02; my thanks to Denise Penrose, Emilia Thiuri, and the rest of the team at Morgan Kaufmann for organizing that event, to the approximately two dozen attendees who shared their thoughts on content and pedagogy, and to the many other individuals who reviewed two subsequent revision plans.

A draft of the second edition was class tested in the fall of 2004 at eight different universities. I am grateful to Gerald Baumgartner (Louisiana State University), William Calhoun (Bloomsburg University), Betty Cheng (Michigan State University), Jingke Li (Portland State University), Beverly Sanders (University of Florida), Darko Stefanovic (University of New Mexico), Raymond Toal (Loyola Marymount University), Robert van Engelen (Florida State University), and all their students for a mountain of suggestions, reactions, bug fixes, and other feedback. Professor van Engelen provided several excellent end-of-chapter exercises.

External reviewers for the second edition also provided a wealth of useful suggestions. My thanks to Richard J. Botting (California State University, San Bernardino), Kamal Dahbur (DePaul University), Stephen A. Edwards (Columbia University), Eileen Head (Binghamton University), Li Liao (University of Delaware), Christopher Vickery (Queens College, City University of New York), Garrett Wollman (MIT), Neng-Fa Zhou (Brooklyn College, City University of New York), and Cynthia Brown Zickos (University of Mississippi). Garrett Wollman's technical review of Chapter 13 was particularly helpful, as were his earlier comments on a variety of topics in the first edition. Sadly, time has not permitted me to do justice to everyone's suggestions. I have incorporated as much as I could, and have carefully saved the rest for guidance on the third edition. Problems that remain in the current edition are entirely my own.

PLP-2e was also class tested at the University of Rochester in the fall of 2004. I am grateful to all my students, and to John Heidkamp, David Lu, and Dan Mullowney in particular, for their enthusiasm and suggestions. Mike Spear provided several helpful pointers on web technology for Chapter 13. Over the previous several years, my colleagues Chen Ding and Sandhya Dwarkadas taught from the first edition several times and had many helpful suggestions. Chen's feedback on Chapter 15 (assisted by Yutao Zhong) was particularly valuable. My thanks as well to the rest of my colleagues, to department chair Mitsunori Ogihara, and to the department's administrative, secretarial, and technical staff for providing such a supportive and productive work environment.

As they were on the first edition, the staff at Morgan Kaufmann have been a genuine pleasure to work with, on both a professional and a personal level. My thanks in particular to Denise Penrose, publisher; Nate McFadden, editor; Carl Soares, production editor; Peter Ashenden, CD designer; Brian Grimm, marketing manager; and Valerie Witte, editorial assistant.

Most important, I am indebted to my wife, Kelly, and our daughters, Erin and Shannon, for their patience and support through endless months of writing and revising. Computing is a fine profession, but family is what really matters.

Michael L. Scott
Rochester, NY
April 2005