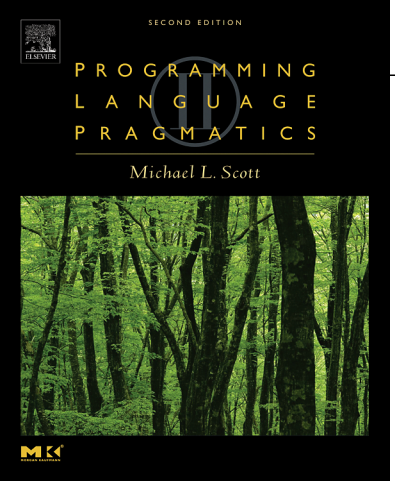
SECOND EDITION
PROGRAMMING
LANGUAGE
PRAGMATICS
Michael L. Scott

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