

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



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.

© 2006 by Elsevier Inc. All rights reserved.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, scanning, or otherwise—without prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com. You may also complete your request on-line via the Elsevier homepage (<http://elsevier.com>) by selecting "Customer Support" and then "Obtaining Permissions."

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

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

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

Foreword	ix
Preface	xxiii

FOUNDATIONS

I Introduction	3
I.1 The Art of Language Design	5
I.2 The Programming Language Spectrum	8
I.3 Why Study Programming Languages?	11
I.4 Compilation and Interpretation	13
I.5 Programming Environments	21
I.6 An Overview of Compilation	22
I.6.1 Lexical and Syntax Analysis	23
I.6.2 Semantic Analysis and Intermediate Code Generation	25
I.6.3 Target Code Generation	28
I.6.4 Code Improvement	30
I.7 Summary and Concluding Remarks	31
I.8 Exercises	32
I.9 Explorations	33
I.10 Bibliographic Notes	35

2 Programming Language Syntax	37
2.1 Specifying Syntax	38
2.1.1 Tokens and Regular Expressions	39
2.1.2 Context-Free Grammars	42
2.1.3 Derivations and Parse Trees	43
2.2 Scanning	46
2.2.1 Generating a Finite Automaton	49
2.2.2 Scanner Code	54
2.2.3 Table-Driven Scanning	58
2.2.4 Lexical Errors	58
2.2.5 Pragmas	60
2.3 Parsing	61
2.3.1 Recursive Descent	64
2.3.2 Table-Driven Top-Down Parsing	70
2.3.3 Bottom-Up Parsing	80
2.3.4 Syntax Errors	CD I • 93
2.4 Theoretical Foundations	CD 13 • 94
2.4.1 Finite Automata	CD 13
2.4.2 Push-Down Automata	CD 16
2.4.3 Grammar and Language Classes	CD 17
2.5 Summary and Concluding Remarks	95
2.6 Exercises	96
2.7 Explorations	101
2.8 Bibliographic Notes	101
3 Names, Scopes, and Bindings	103
3.1 The Notion of Binding Time	104
3.2 Object Lifetime and Storage Management	106
3.2.1 Static Allocation	107
3.2.2 Stack-Based Allocation	109
3.2.3 Heap-Based Allocation	111
3.2.4 Garbage Collection	113
3.3 Scope Rules	114
3.3.1 Static Scope	115
3.3.2 Nested Subroutines	117
3.3.3 Declaration Order	119
3.3.4 Modules	124

3.3.5 Module Types and Classes	128
3.3.6 Dynamic Scope	131
3.4 Implementing Scope	CD 23 · 135
3.4.1 Symbol Tables	CD 23
3.4.2 Association Lists and Central Reference Tables	CD 27
3.5 The Binding of Referencing Environments	136
3.5.1 Subroutine Closures	138
3.5.2 First- and Second-Class Subroutines	140
3.6 Binding Within a Scope	142
3.6.1 Aliases	142
3.6.2 Overloading	143
3.6.3 Polymorphism and Related Concepts	145
3.7 Separate Compilation	CD 30 · 149
3.7.1 Separate Compilation in C	CD 30
3.7.2 Packages and Automatic Header Inference	CD 33
3.7.3 Module Hierarchies	CD 35
3.8 Summary and Concluding Remarks	149
3.9 Exercises	151
3.10 Explorations	157
3.11 Bibliographic Notes	158
4 Semantic Analysis	161
4.1 The Role of the Semantic Analyzer	162
4.2 Attribute Grammars	166
4.3 Evaluating Attributes	168
4.4 Action Routines	179
4.5 Space Management for Attributes	CD 39 · 181
4.5.1 Bottom-Up Evaluation	CD 39
4.5.2 Top-Down Evaluation	CD 44
4.6 Decorating a Syntax Tree	182
4.7 Summary and Concluding Remarks	187
4.8 Exercises	189
4.9 Explorations	193
4.10 Bibliographic Notes	194

5 Target Machine Architecture	195
5.1 The Memory Hierarchy	196
5.2 Data Representation	199
5.2.1 Computer Arithmetic	CD 54 · 199
5.3 Instruction Set Architecture	201
5.3.1 Addressing Modes	201
5.3.2 Conditions and Branches	202
5.4 Architecture and Implementation	204
5.4.1 Microprogramming	205
5.4.2 Microprocessors	206
5.4.3 RISC	207
5.4.4 Two Example Architectures: The x86 and MIPS	CD 59 · 208
5.4.5 Pseudo-Assembly Notation	209
5.5 Compiling for Modern Processors	210
5.5.1 Keeping the Pipeline Full	211
5.5.2 Register Allocation	216
5.6 Summary and Concluding Remarks	221
5.7 Exercises	223
5.8 Explorations	226
5.9 Bibliographic Notes	227

II CORE ISSUES IN LANGUAGE DESIGN 231

6 Control Flow	233
6.1 Expression Evaluation	234
6.1.1 Precedence and Associativity	236
6.1.2 Assignments	238
6.1.3 Initialization	246
6.1.4 Ordering Within Expressions	249
6.1.5 Short-Circuit Evaluation	252
6.2 Structured and Unstructured Flow	254
6.2.1 Structured Alternatives to <code>goto</code>	255
6.2.2 Continuations	259

6.3 Sequencing	260
6.4 Selection	261
6.4.1 Short-Circuited Conditions	262
6.4.2 Case/Switch Statements	265
6.5 Iteration	270
6.5.1 Enumeration-Controlled Loops	271
6.5.2 Combination Loops	277
6.5.3 Iterators	278
6.5.4 Generators in Icon	CD 69 · 284
6.5.5 Logically Controlled Loops	284
6.6 Recursion	287
6.6.1 Iteration and Recursion	287
6.6.2 Applicative- and Normal-Order Evaluation	291
6.7 Nondeterminacy	CD 72 · 295
6.8 Summary and Concluding Remarks	296
6.9 Exercises	298
6.10 Explorations	304
6.11 Bibliographic Notes	305
7 Data Types	307
7.1 Type Systems	308
7.1.1 Type Checking	309
7.1.2 Polymorphism	309
7.1.3 The Definition of Types	311
7.1.4 The Classification of Types	312
7.1.5 Orthogonality	319
7.2 Type Checking	321
7.2.1 Type Equivalence	321
7.2.2 Type Compatibility	327
7.2.3 Type Inference	332
7.2.4 The ML Type System	CD 81 · 335
7.3 Records (Structures) and Variants (Unions)	336
7.3.1 Syntax and Operations	337
7.3.2 Memory Layout and Its Impact	338
7.3.3 With Statements	CD 90 · 341
7.3.4 Variant Records	341

7.4 Arrays	349
7.4.1 Syntax and Operations	349
7.4.2 Dimensions, Bounds, and Allocation	353
7.4.3 Memory Layout	358
7.5 Strings	366
7.6 Sets	367
7.7 Pointers and Recursive Types	369
7.7.1 Syntax and Operations	370
7.7.2 Dangling References	379
7.7.3 Garbage Collection	383
7.8 Lists	389
7.9 Files and Input/Output	CD 93 · 392
7.9.1 Interactive I/O	CD 93
7.9.2 File-Based I/O	CD 94
7.9.3 Text I/O	CD 96
7.10 Equality Testing and Assignment	393
7.11 Summary and Concluding Remarks	395
7.12 Exercises	398
7.13 Explorations	404
7.14 Bibliographic Notes	405
8 Subroutines and Control Abstraction	407
8.1 Review of Stack Layout	408
8.2 Calling Sequences	410
8.2.1 Displays	CD 107 · 413
8.2.2 Case Studies: C on the MIPS; Pascal on the x86	CD 111 · 414
8.2.3 Register Windows	CD 119 · 414
8.2.4 In-Line Expansion	415
8.3 Parameter Passing	417
8.3.1 Parameter Modes	418
8.3.2 Call by Name	CD 122 · 426
8.3.3 Special Purpose Parameters	427
8.3.4 Function Returns	432
8.4 Generic Subroutines and Modules	434
8.4.1 Implementation Options	435
8.4.2 Generic Parameter Constraints	437

8.4.3 Implicit Instantiation	440
8.4.4 Generics in C++, Java, and C#	CD 125 • 440
8.5 Exception Handling	441
8.5.1 Defining Exceptions	443
8.5.2 Exception Propagation	445
8.5.3 Example: Phrase-Level Recovery in a Recursive Descent Parser	448
8.5.4 Implementation of Exceptions	449
8.6 Coroutines	453
8.6.1 Stack Allocation	455
8.6.2 Transfer	457
8.6.3 Implementation of Iterators	CD 135 • 458
8.6.4 Discrete Event Simulation	CD 139 • 458
8.7 Summary and Concluding Remarks	459
8.8 Exercises	460
8.9 Explorations	466
8.10 Bibliographic Notes	467
9 Data Abstraction and Object Orientation	469
9.1 Object-Oriented Programming	471
9.2 Encapsulation and Inheritance	481
9.2.1 Modules	481
9.2.2 Classes	484
9.2.3 Type Extensions	486
9.3 Initialization and Finalization	489
9.3.1 Choosing a Constructor	490
9.3.2 References and Values	491
9.3.3 Execution Order	495
9.3.4 Garbage Collection	496
9.4 Dynamic Method Binding	497
9.4.1 Virtual and Nonvirtual Methods	500
9.4.2 Abstract Classes	501
9.4.3 Member Lookup	502
9.4.4 Polymorphism	505
9.4.5 Closures	508
9.5 Multiple Inheritance	CD 146 • 511
9.5.1 Semantic Ambiguities	CD 148

9.5.2 Replicated Inheritance	CD 151
9.5.3 Shared Inheritance	CD 152
9.5.4 Mix-In Inheritance	CD 154
9.6 Object-Oriented Programming Revisited	512
9.6.1 The Object Model of Smalltalk	CD 158 · 513
9.7 Summary and Concluding Remarks	513
9.8 Exercises	515
9.9 Explorations	517
9.10 Bibliographic Notes	518



ALTERNATIVE PROGRAMMING MODELS

521

10 Functional Languages	523
10.1 Historical Origins	524
10.2 Functional Programming Concepts	526
10.3 A Review/Overview of Scheme	528
10.3.1 Bindings	530
10.3.2 Lists and Numbers	531
10.3.3 Equality Testing and Searching	532
10.3.4 Control Flow and Assignment	533
10.3.5 Programs as Lists	535
10.3.6 Extended Example: DFA Simulation	537
10.4 Evaluation Order Revisited	539
10.4.1 Strictness and Lazy Evaluation	541
10.4.2 I/O: Streams and Monads	542
10.5 Higher-Order Functions	545
10.6 Theoretical Foundations	CD 166 · 549
10.6.1 Lambda Calculus	CD 168
10.6.2 Control Flow	CD 171
10.6.3 Structures	CD 173
10.7 Functional Programming in Perspective	549
10.8 Summary and Concluding Remarks	552

10.9 Exercises	552
10.10 Explorations	557
10.11 Bibliographic Notes	558
11 Logic Languages	559
11.1 Logic Programming Concepts	560
11.2 Prolog	561
11.2.1 Resolution and Unification	563
11.2.2 Lists	564
11.2.3 Arithmetic	565
11.2.4 Search/Execution Order	566
11.2.5 Extended Example: Tic-Tac-Toe	569
11.2.6 Imperative Control Flow	571
11.2.7 Database Manipulation	574
11.3 Theoretical Foundations	CD 180 · 579
11.3.1 Clausal Form	CD 181
11.3.2 Limitations	CD 182
11.3.3 Skolemization	CD 183
11.4 Logic Programming in Perspective	579
11.4.1 Parts of Logic Not Covered	580
11.4.2 Execution Order	580
11.4.3 Negation and the “Closed World” Assumption	581
11.5 Summary and Concluding Remarks	583
11.6 Exercises	584
11.7 Explorations	586
11.8 Bibliographic Notes	587
12 Concurrency	589
12.1 Background and Motivation	590
12.1.1 A Little History	590
12.1.2 The Case for Multithreaded Programs	593
12.1.3 Multiprocessor Architecture	597
12.2 Concurrent Programming Fundamentals	601
12.2.1 Communication and Synchronization	601
12.2.2 Languages and Libraries	603
12.2.3 Thread Creation Syntax	604

12.2.4 Implementation of Threads	613
12.3 Shared Memory	619
12.3.1 Busy-Wait Synchronization	620
12.3.2 Scheduler Implementation	623
12.3.3 Semaphores	627
12.3.4 Monitors	629
12.3.5 Conditional Critical Regions	634
12.3.6 Implicit Synchronization	638
12.4 Message Passing	642
12.4.1 Naming Communication Partners	642
12.4.2 Sending	646
12.4.3 Receiving	651
12.4.4 Remote Procedure Call	656
12.5 Summary and Concluding Remarks	660
12.6 Exercises	662
12.7 Explorations	668
12.8 Bibliographic Notes	669
13 Scripting Languages	671
13.1 What Is a Scripting Language?	672
13.1.1 Common Characteristics	674
13.2 Problem Domains	677
13.2.1 Shell (Command) Languages	677
13.2.2 Text Processing and Report Generation	684
13.2.3 Mathematics and Statistics	689
13.2.4 “Glue” Languages and General Purpose Scripting	690
13.2.5 Extension Languages	698
13.3 Scripting the World Wide Web	701
13.3.1 CGI Scripts	702
13.3.2 Embedded Server-Side Scripts	703
13.3.3 Client-Side Scripts	708
13.3.4 Java Applets	708
13.3.5 XSLT	712
13.4 Innovative Features	722
13.4.1 Names and Scopes	723
13.4.2 String and Pattern Manipulation	728
13.4.3 Data Types	736

13.4.4 Object Orientation	741
13.5 Summary and Concluding Remarks	748
13.6 Exercises	750
13.7 Explorations	755
13.8 Bibliographic Notes	756

IV A CLOSER LOOK AT IMPLEMENTATION 759

14 Building a Runnable Program	761
14.1 Back-End Compiler Structure	761
14.1.1 A Plausible Set of Phases	762
14.1.2 Phases and Passes	766
14.2 Intermediate Forms	CD 189 • 766
14.2.1 Diana	CD 189
14.2.2 GNU RTL	CD 192
14.3 Code Generation	769
14.3.1 An Attribute Grammar Example	769
14.3.2 Register Allocation	772
14.4 Address Space Organization	775
14.5 Assembly	776
14.5.1 Emitting Instructions	778
14.5.2 Assigning Addresses to Names	780
14.6 Linking	781
14.6.1 Relocation and Name Resolution	782
14.6.2 Type Checking	783
14.7 Dynamic Linking	CD 195 • 784
14.7.1 Position-Independent Code	CD 195
14.7.2 Fully Dynamic (Lazy) Linking	CD 196
14.8 Summary and Concluding Remarks	786
14.9 Exercises	787
14.10 Explorations	789
14.11 Bibliographic Notes	790

15 Code Improvement	CD 202 · 791
15.1 Phases of Code Improvement	CD 204
15.2 Peephole Optimization	CD 206
15.3 Redundancy Elimination in Basic Blocks	CD 209
15.3.1 A Running Example	CD 210
15.3.2 Value Numbering	CD 211
15.4 Global Redundancy and Data Flow Analysis	CD 217
15.4.1 SSA Form and Global Value Numbering	CD 218
15.4.2 Global Common Subexpression Elimination	CD 220
15.5 Loop Improvement I	CD 227
15.5.1 Loop Invariants	CD 228
15.5.2 Induction Variables	CD 229
15.6 Instruction Scheduling	CD 232
15.7 Loop Improvement II	CD 236
15.7.1 Loop Unrolling and Software Pipelining	CD 237
15.7.2 Loop Reordering	CD 241
15.8 Register Allocation	CD 248
15.9 Summary and Concluding Remarks	CD 252
15.10 Exercises	CD 253
15.11 Explorations	CD 257
15.12 Bibliographic Notes	CD 258
A Programming Languages Mentioned	793
B Language Design and Language Implementation	803
C Numbered Examples	807
Bibliography	819
Index	837

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

- [Ulysses for free](#)
- [download Cruising Attitude: Tales of Crashpads, Crew Drama, and Crazy Passengers at 35,000 Feet pdf, azw \(kindle\), epub, doc, mobi](#)
- [Louder Than Hell: The Definitive Oral History of Metal pdf, azw \(kindle\), epub](#)
- [download online Confusion: Roman \(Der Barock-Zyklus, Band 2\) here](#)

- <http://www.celebritychat.in/?ebooks/Ulysses.pdf>
- <http://korplast.gr/lib/Cruising-Attitude--Tales-of-Crashpads--Crew-Drama--and-Crazy-Passengers-at-35-000-Feet.pdf>
- <http://betsy.wesleychapelcomputerrepair.com/library/Louder-Than-Hell--The-Definitive-Oral-History-of-Metal.pdf>
- <http://cambridgebrass.com/?freebooks/Confusion--Roman--Der-Barock-Zyklus--Band-2-.pdf>