Preface

Changes for the Tenth Edition

The goals, overall structure, and approach of this tenth edition of *Concepts of Programming Languages* remain the same as those of the nine earlier editions. The principal goals are to introduce the main constructs of contemporary programming languages and to provide the reader with the tools necessary for the critical evaluation of existing and future programming languages. A secondary goal is to prepare the reader for the study of compiler design, by providing an in-depth discussion of programming language structures, presenting a formal method of describing syntax and introducing approaches to lexical and syntactic analysis.

The tenth edition evolved from the ninth through several different kinds of changes. To maintain the currency of the material, some of the discussion of older programming languages has been removed. For example, the description of COBOL’s record operations was removed from Chapter 6 and that of Fortran’s *do* statement was removed from Chapter 8. Likewise, the description of Ada’s generic subprograms was removed from Chapter 9 and the discussion of Ada’s asynchronous message passing was removed from Chapter 13.

On the other hand, a section on closures, a section on calling subprograms indirectly, and a section on generic functions in F# were added to Chapter 9; sections on Objective-C were added to Chapters 11 and 12; a section on concurrency in functional programming languages was added to Chapter 13; a section on C# event handling was added to Chapter 14; a section on F# and a section on support for functional programming in primarily imperative languages were added to Chapter 15.

In some cases, material has been moved. For example, several different discussions of constructs in functional programming languages were moved from Chapter 15 to earlier chapters. Among these were the descriptions of the control statements in functional programming languages to Chapter 8 and the lists and list operations of Scheme and ML to Chapter 6. These moves indicate a significant shift in the philosophy of the book—in a sense, the mainstreaming of some of the constructs of functional programming languages. In previous editions, all discussions of functional programming language constructs were segregated in Chapter 15.

Chapters 11, 12, and 15 were substantially revised, with five figures being added to Chapter 12.

Finally, numerous minor changes were made to a large number of sections of the book, primarily to improve clarity.
The Vision

This book describes the fundamental concepts of programming languages by discussing the design issues of the various language constructs, examining the design choices for these constructs in some of the most common languages, and critically comparing design alternatives.

Any serious study of programming languages requires an examination of some related topics, among which are formal methods of describing the syntax and semantics of programming languages, which are covered in Chapter 3. Also, implementation techniques for various language constructs must be considered: Lexical and syntax analysis are discussed in Chapter 4, and implementation of subprogram linkage is covered in Chapter 10. Implementation of some other language constructs is discussed in various other parts of the book.

The following paragraphs outline the contents of the tenth edition.

Chapter Outlines

Chapter 1 begins with a rationale for studying programming languages. It then discusses the criteria used for evaluating programming languages and language constructs. The primary influences on language design, common design trade-offs, and the basic approaches to implementation are also examined.

Chapter 2 outlines the evolution of most of the important languages discussed in this book. Although no language is described completely, the origins, purposes, and contributions of each are discussed. This historical overview is valuable, because it provides the background necessary to understanding the practical and theoretical basis for contemporary language design. It also motivates further study of language design and evaluation. In addition, because none of the remainder of the book depends on Chapter 2, it can be read on its own, independent of the other chapters.

Chapter 3 describes the primary formal method for describing the syntax of programming language—BNF. This is followed by a description of attribute grammars, which describe both the syntax and static semantics of languages. The difficult task of semantic description is then explored, including brief introductions to the three most common methods: operational, denotational, and axiomatic semantics.

Chapter 4 introduces lexical and syntax analysis. This chapter is targeted to those colleges that no longer require a compiler design course in their curricula. Like Chapter 2, this chapter stands alone and can be read independently of the rest of the book.

Chapters 5 through 14 describe in detail the design issues for the primary constructs of programming languages. In each case, the design choices for several example languages are presented and evaluated. Specifically, Chapter 5 covers the many characteristics of variables, Chapter 6 covers data types, and Chapter 7 explains expressions and assignment statements. Chapter 8 describes control
statements, and Chapters 9 and 10 discuss subprograms and their implementa-
tion. Chapter 11 examines data abstraction facilities. Chapter 12 provides an in-
depth discussion of language features that support object-oriented programming
(inheritance and dynamic method binding), Chapter 13 discusses concurrent
program units, and Chapter 14 is about exception handling, along with a brief
discussion of event handling.

The last two chapters (15 and 16) describe two of the most important alterna-
tive programming paradigms: functional programming and logic programming.
However, some of the data structures and control constructs of functional pro-
gramming languages are discussed in Chapters 6 and 8. Chapter 15 presents an
introduction to Scheme, including descriptions of some of its primitive functions,
special forms, and functional forms, as well as some examples of simple func-
tions written in Scheme. Brief introductions to ML, Haskell, and F# are given
to illustrate some different directions in functional language design. Chapter 16
introduces logic programming and the logic programming language, Prolog.

To the Instructor

In the junior-level programming language course at the University of Colorado
at Colorado Springs, the book is used as follows: We typically cover Chapters 1
and 3 in detail, and though students find it interesting and beneficial reading,
Chapter 2 receives little lecture time due to its lack of hard technical content.
Because no material in subsequent chapters depends on Chapter 2, as noted
earlier, it can be skipped entirely, and because we require a course in compiler
design, Chapter 4 is not covered.

Chapters 5 through 9 should be relatively easy for students with extensive
programming experience in C++, Java, or C#. Chapters 10 through 14 are more
challenging and require more detailed lectures.

Chapters 15 and 16 are entirely new to most students at the junior level.
Ideally, language processors for Scheme and Prolog should be available for
students required to learn the material in these chapters. Sufficient material is
included to allow students to dabble with some simple programs.

Undergraduate courses will probably not be able to cover all of the mate-
rial in the last two chapters. Graduate courses, however, should be able to
completely discuss the material in those chapters by skipping over parts of the
early chapters on imperative languages.

Supplemental Materials

The following supplements are available to all readers of this book at www.
pearsonhighered.com/cssupport.

- A set of lecture note slides. PowerPoint slides are available for each chapter
  in the book.
- PowerPoint slides containing all the figures in the book.

A companion Website to the book is available at www.pearsonhighered.com/sebesta. This site contains mini-manuals (approximately 100-page tutorials) on a handful of languages. These proceed on the assumption that the student knows how to program in some other language, giving the student enough information to complete the chapter materials in each language. Currently the site includes manuals for C++, C, Java, and Smalltalk.

Solutions to many of the problem sets are available to qualified instructors in our Instructor Resource Center at www.pearsonhighered.com/irc. Please contact your school’s Pearson Education representative or visit www.pearsonhighered.com/irc to register.

**Language Processor Availability**

Processors for and information about some of the programming languages discussed in this book can be found at the following Websites:

- C, C++, Fortran, and Ada: gcc.gnu.org
- C# and F#: microsoft.com
- Java: java.sun.com
- Haskell: haskell.org
- Lua: www.lua.org
- Scheme: www.plt-scheme.org/software/drscheme
- Perl: www.perl.com
- Python: www.python.org
- Ruby: www.ruby-lang.org

JavaScript is included in virtually all browsers; PHP is included in virtually all Web servers.

All this information is also included on the companion Website.

**Acknowledgments**

The suggestions from outstanding reviewers contributed greatly to this book’s present form. In alphabetical order, they are:

Matthew Michael Burke  
I-ping Chu  
Teresa Cole  
Pamela Cutter  
Amer Diwan  
Stephen Edwards  
David E. Goldschmidt  
Nigel Gwee

DePaul University  
Boise State University  
Kalamazoo College  
University of Colorado  
Virginia Tech  
Southern University–Baton Rouge
Preface

Timothy Henry  
University of Rhode Island

Paul M. Jackowitz  
University of Scranton

Duane J. Jarc  
University of Maryland, University College

K. N. King  
Georgia State University

Donald Kraft  
Louisiana State University

Simon H. Lin  
California State University–Northridge

Mark Llewellyn  
University of Central Florida

Bruce R. Maxim  
University of Michigan–Dearborn

Robert McCloskey  
University of Scranton

Curtis Meadow  
University of Maine

Gloria Melara  
California State University–Northridge

Frank J. Mitropoulos  
Nova Southeastern University

Euripides Montagne  
University of Central Florida

Serita Nelesen  
Calvin College

Bob Neufeld  
Wichita State University

Charles Nicholas  
University of Maryland–Baltimore County

Tim R. Norton  
University of Colorado–Colorado Springs

Richard M. Osborne  
University of Colorado–Denver

Saverio Perugini  
University of Dayton

Walter Pharr  
College of Charleston

Michael Prentice  
SUNY Buffalo

Amar Raheja  
California State Polytechnic University–Pomona

Hossein Saiedian  
University of Kansas

Stuart C. Shapiro  
SUNY Buffalo

Neelam Soundarajan  
Ohio State University

Ryan Stansifer  
Florida Institute of Technology

Nancy Tinkham  
Rowan University

Paul Tymann  
Rochester Institute of Technology

Cristian Videira Lopes  
University of California–Irvine

Sumanth Yenduri  
University of Southern Mississippi

Salih Yurttas  
Texas A&M University

Numerous other people provided input for the previous editions of Concepts of Programming Languages at various stages of its development. All of their comments were useful and greatly appreciated. In alphabetical order, they are: Vicki Allan, Henry Bauer, Carter Bays, Manuel E. Bermudez, Peter Brouwer, Margaret Burnett, Paosheng Chang, Liang Cheng, John Crenshaw, Charles Dana, Barbara Ann Griem, Mary Lou Haag, John V. Harrison, Eileen Head, Ralph C. Hilzer, Eric Joanis, Leon Jololian, Hikyoo Koh, Jiang B. Liu, Meiliu Lu, Jon Mauney, Robert McCoard, Dennis L. Mumaugh, Michael G. Murphy, Andrew Oldroyd, Young Park, Rebecca Parsons, Steve J. Phelps, Jeffery Popyack, Raghvinder Sangwan, Steven Rapkin, Hamilton Richard, Tom Sager, Joseph Schell, Sibylle Schupp, Mary Louise Soffa, Neelam Soundarajan, Ryan Stansifer, Steve Stevenson, Virginia Teller, Yang Wang, John M. Weiss, Franck Xia, and Salih Yurnas.
Matt Goldstein, editor; Chelsea Kharakozova, editorial assistant; and, Marilyn Lloyd, senior production manager of Addison-Wesley, and Gillian Hall of The Aardvark Group Publishing Services, all deserve my gratitude for their efforts to produce the tenth edition both quickly and carefully.

About the Author

Robert Sebesta is an Associate Professor Emeritus in the Computer Science Department at the University of Colorado–Colorado Springs. Professor Sebesta received a BS in applied mathematics from the University of Colorado in Boulder and MS and PhD degrees in computer science from Pennsylvania State University. He has taught computer science for more than 38 years. His professional interests are the design and evaluation of programming languages.