国外计算机科学教材系列

程序设计语言
——原理与实践（第二版）

Programming Languages
Principles and Practice
Second Edition

英文版

[美] Kenneth C. Louden 著

THOMSON BROOKS/COLE

電子工業出版社
Publishing House of Electronics Industry
www.phei.com.cn

国外计算机科学教材系列

程序设计语言——原理与实践

（第二版）

（英文版）

Programming Languages

Principles and Practice

Second Edition

［美］ Kenneth C. Louden 著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书介绍了程序设计语言的一般概念，包括程序设计语言的语法和语义，涉及命令式语言、面向对象语言、函数式语言、逻辑式语言和并行语言等多种范例，分析了各种语言的设计原理和内在机制，讨论了语言的理论基础和实现时必须考虑的问题。

本书可用于计算机及其相关专业学生的双语教材，软件与理论专业研究生相关课程的参考书，也可供计算机专业人员参考。

981-243-498-4

版权贸易合同登记号：图字：01-2002-4545

图书在版编目（CIP）数据

程序设计语言——原理与实践=Programming Languages: Principles and Practice（第二版）/（美）劳登（Louden, K. C.）著. -北京：电子工业出版社，2003.1
（国外计算机科学教材系列）
ISBN 7-5053-8261-6

Ⅰ.程... Ⅱ.劳... Ⅲ.程序语言－英文 Ⅳ. TP312

中国版本图书馆CIP数据核字（2002）第092807号

责任编辑：李秦华
印 刷 者：北京天宇星印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编：100036
经 销：各地新华书店
开 本：787 × 980 1/16 印张：44.75 字数：1146千字
版 次：2003年1月第1版 2003年1月第1次印刷
定 价：58.00元

出 版 说 明

21世纪初的5至10年是我国国民经济和社会发展的重要时期，也是信息产业快速发展的关键时期。在我国加入WTO后的今天，培养一支适应国际化竞争的一流IT人才队伍是我国高等教育的重要任务之一。信息科学和技术方面人才的优劣与多寡，是我国面对国际竞争时成败的关键因素。

当前，正值我国高等教育特别是信息科学领域的教育调整、变革的重大时期，为使我国教育体制与国际化接轨，有条件的高等院校正在为某些信息学科和技术课程使用国外优秀教材和优秀原版教材，以使我国在计算机教学上尽快赶上国际先进水平。

电子工业出版社秉承多年来引进国外优秀图书的经验，翻译出版了“国外计算机科学教材系列”丛书，这套教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。这些教材涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。同时，我们也适当引进了一些优秀英文原版教材，本着翻译版本和英文原版并重的原则，对重点图书既提供英文原版又提供相应的翻译版本。

在图书选题上，我们大都选择国外著名出版公司出版的高校教材，如Pearson Education培生教育出版集团、麦格劳－希尔教育出版集团、麻省理工学院出版社、剑桥大学出版社等。撰写教材的许多作者都是蜚声世界的教授、学者，如道格拉斯·科默（Douglas E. Comer）、威廉·斯托林斯（William Stallings）、哈维·戴特尔（Harvey M. Deitel）、尤利斯·布莱克（Uyless Black）等。

为确保教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本系列教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师、博士，也有积累了几十年教学经验的老教授和博士生导师。

在该系列教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括对所选教材进行全面论证；选择编辑时力求达到专业对口；对排版、印制质量进行严格把关。对于英文教材中出现的错误，我们通过与作者联络和网上下载勘误表等方式，逐一进行了修订。

此外，我们还将与国外著名出版公司合作，提供一些教材的教学支持资料，希望能为授课老师提供帮助。今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

电子工业出版社

教材出版委员会

Preface

This book is an introduction to the broad field of programming languages. It combines a general presentation of principles with considerable detail about many modern languages, including some of the newest functional and object-oriented languages. Unlike many introductory texts, it contains significant material on implementation issues, the theoretical foundations of programming languages, and a large number of exercises. All of these features make this text a useful bridge to compiler courses and to the theoretical study of programming languages. However, it is a text specifically designed for an advanced undergraduate programming languages survey course that covers most of the programming languages requirements specified in the 2001 ACM/IEEE-CS Joint Curriculum Task Force Report, and the CS8 course of the 1978 ACM Curriculum.

My goals in preparing this new edition are to bring the language-specific material in line with the changes in the popularity and use of programming languages since the publication of the first edition in 1993, to improve and expand the coverage in certain areas, and to improve the presentation and usefulness of the examples and exercises, while retaining as much of the original text and organization as possible.

Students are not expected to know any one particular language. However, experience with at least one language is necessary. A certain degree of "computational sophistication," such as that provided by a course in data structures and a discrete mathematics course, is also expected. Major languages used in this edition include C, C++, Java, Ada, ML, Haskell, Scheme, and Prolog; many other languages are discussed more briefly.

Overview and Organization

In most cases, each chapter largely is independent of the others without artificially restricting the material in each. Cross references in the text allow the student or instructor to fill in any gaps that might arise even if a particular chapter or section is skipped.

Chapter 1 surveys the concepts studied in later chapters, and introduces the different language paradigms with simple examples in typical languages.

Chapters 2 and 3 provide overviews of the history of programming languages and language design principles, respectively. Chapter 3 could serve well as a culminating chapter for the book, but I find it arouses interest in later topics when covered here.

Chapter 4 treats syntax in some detail, including the use of BNF, EBNF, and syntax diagrams. A brief section views recursive definitions (like BNF) as set equations to be solved, a view that recurs periodically throughout the text. One section is devoted to recursive-descent parsing and the use of parsing tools.

Chapters 5, 6, 7, and 8 cover the central semantic issues of programming languages: declaration, allocation, evaluation; the symbol table and runtime environment as semantic functions; data types and type checking; procedure activation and parameter passing; and exceptions and exception handling.

Chapter 9 gives an overview of modules and abstract data types, including language mechanisms and equational, or algebraic, specification.

Chapters 10, 11, 12, and 13 address language paradigms, beginning with the object-oriented paradigms in Chapter 10. I use Java to introduce the concepts in this chapter. Individual sections feature C++ and Smalltalk. Chapter 11 deals with the functional paradigm. Each of the languages Scheme, ML, and Haskell are covered in some detail. This chapter also introduces the lambda calculus and the theory of recursive function definitions. Chapter 12, on logic programming, offers an extended section on Prolog, and devotes one section to equational languages such as OBJ.

Chapter 13 introduces the three principal methods of formal semantics: operational, denotational, and axiomatic. This is somewhat unique among introductory texts in that it gives enough detail to provide a real flavor for the methods.

Chapter 14 treats the major ways parallelism has been introduced into programming languages: coroutines, threads, semaphores, monitors, and message passing, with examples primarily from Java and Ada. Its final section surveys recent efforts to introduce parallelism into LISP and Prolog.

Use as a Text

I have used this text for more than ten years in my CS 152 classes of upper division computer science majors and graduate students at San Jose State University. I have taught the course using two completely different organizations, which could loosely be called the "principles" approach and the "paradigm" approach. Two suggested organizations of these approaches in a semester-long course are as follow:

> *The principles approach*: Chapters 1, 4, 5, 6, 7, 8, and 9. If there is extra time, Chapters 2 and 3.

The paradigm approach: Chapters 1, 10, 11, 12, 13, and 14 (not necessarily in that order). If there is extra time, Chapters 2 and 3, or selected topics from the remaining chapters.

In a two-semester or two-quarter sequence it should be possible to cover most of the book.

Selected answers for many of the exercises at the end of each chapter may be found at www.brookscole.com or on the author's Web site, www.cs.sjsu.edu/faculty/louden. Many are programming exercises (none extremely long) in languages discussed in the text. Conceptual exercises range from the short-answer type that test understanding of the material to longer, essay-style exercises and challenging "thought" questions. A few moments' reflection should give the reader adequate insight into the potential difficulty of a particular exercise. Further knowledge can be gained by reading the on-line answers, which I treat as an extension of the text and sometimes provide additional information beyond that required to solve the problem. Occasionally the answer to an exercise on a particular language requires the reader to consult a language reference manual or have knowledge of the language not specifically covered in the text. Throughout the book I have tried to improve the usefulness of the code examples by adding line numbers where appropriate, and by augmenting many examples with main program drivers that allow them to be executed to demonstrate their described behavior. All such examples, as well as a number of others (in which, for space or other reasons, such extra code was suppressed), are available through www.brookscole.com or the author's Web site listed above. These Web sites also contain links to free, downloadable translators for all the major languages of the book, many of which I have used to test the examples. Other materials may also be available.

Summary of Changes between the First and Second Editions

In the first edition, I used examples from the most widely known imperative languages, including C, Pascal, Ada, Modula-2, and FORTRAN, as well as some of the less widely known languages representing other language paradigms, such as Scheme, ML, Miranda, C++, Eiffel, Smalltalk, and Prolog. The most extensive change in the current edition is the replacement of Pascal and Modula-2 largely by C, C++, and Java in the examples. Modula-2 has disappeared, except for a "historical" section in Chapter 9, on ADTs; a few examples in Pascal remain. I also use Ada quite a bit, especially for features that are not well represented in C/C++/Java (e.g., subranges, arrays and slices, name equivalence of data types). Java replaces Simula as the primary example in Chapter 10, on object-oriented programming languages, and I eliminated the section on Eiffel. I devote considerably more space to ML and Haskell in Chapter 11, on functional languages, and I added ML examples liberally throughout the book. Finally, I use Java threads as the basic example of concurrency

in Chapter 14, on parallel programming languages. Additional significant changes are as follows:

- I split off procedures and environments from the rest of the control material, so Chapter 7 now treats control expressions and statements, and the new Chapter 8 treats control procedures and environments. I moved expressions from the end of Chapter 5, on basic semantics, to the beginning of Chapter 7. Because in most cases some implicit or explicit control is inherent in evaluating expressions, this topic fits well with other control issues.
- I include overloading with the symbol table material in Chapter 5, because it essentially is a symbol table task to disambiguate overloaded identifiers. While this presents a "phase order" problem with Chapter 6, on data types—the type signature being the primary attribute used in overload resolution—the amount of data type information needed to understand overload resolution is not great, and the material seems more natural presented in this way.
- I include parametric polymorphism with the discussion of type checking in Chapter 6, in which I also give a more extensive account of Hindley-Milner polymorphic type checking. Parametric polymophism comes up again in Chapter 9 in discussing Ada packages, and in Chapter 10 in discussing C++ class templates.
- I rewrote Chapter 9 on ADTs and modules to emphasize modules a bit more and changed its title to include modules. This topic is more challenging than most to present concisely, because the design of ADT and module mechanisms differs more widely among common languages than any other feature except, possibly, concurrency mechanisms. I use ML and Ada as the major examples here, with some additional material on C++ namespaces and Java packages. I defer the use of classes to represent ADTs and modules to Chapter 10 on object-oriented programming.
- I do not mention the scripting languages, such as Perl, JavaScript, and Tcl, extensively in this text (except for a brief section in Chapter 2). While the use of such languages is widespread and increasing, particularly for Web applications, and student interest in them is intense, I still consider them somewhat too special-purpose for this text. However, nothing would prevent the interested instructor from providing examples in these languages of virtually every major language feature.
- I also do not cover any "visual" languages or component assembly tools, such as Visual Basic or various JavaBean tools. My view is that these "languages" are better studied in a GUI or software engineering course. Similarly, I only mention the various markup languages such as XML, SGML, and HTML, in passing.

Acknowledgments

I would like to thank all those persons too numerous to mention who, over the years, have emailed me with comments, corrections, and suggestions. I especially thank the reviewers of this edition for their many useful suggestions and comments: Leonard M. Faltz of Arizona State University, Jesse Yu of the College of St. Elizabeth, Mike Frazier of Abilene Christian University, Ariel Ortiz Ramírez of ITESM Campus Estado de Mexico, James J. Ball of Indiana State University, Samuel A. Rebelsky of Grinnell College, and Arthur Fleck of the University of Iowa.

I also thank Eran Yahav of Tel-Aviv University for reading and commenting on Chapter 14 on concurrency, and Hamilton Richards of the University of Texas for his comments and suggestions on survey Chapter 1 and Chapter 11, on functional programming.

I remain grateful to the many students in my CS 152 sections at San Jose State University for their direct and indirect contributions to this edition and to the previous edition; to my colleagues at San Jose State, Michael Beeson, Cay Horstmann, and Vinh Phat, who read and commented on individual chapters in the first edition; and to the reviewers of that edition, Ray Fanselau of American River College, Larry Irwin of Ohio University, Zane C. Motteler of California Polytechnic State University, Tony P. Ng of the University of Illinois-Urbana, Rick Ruth of Shippensburg University of Pennsylvania, and Ryan Stansifer of the University of North Texas.

Of course, I alone am responsible for the shortcomings and errors in this book. I am happy to receive reports of errors and any other comments from readers at louden@cs.sjsu.edu.

I particularly thank Kallie Swanson, computer science editor at Brooks/Cole, for her encouragement and patience during the seemingly endless process of revision. A special thanks is owed to Marjorie Schlaikjer, who first convinced me to write this book.

Finally, I give my thanks and appreciation, as ever, for the patience, support, and love of my wife, Margreth, and my sons, Andrew and Robin. Without you, much of this would never have happened.

目录概览

Contents