经 典 原 版 书 库

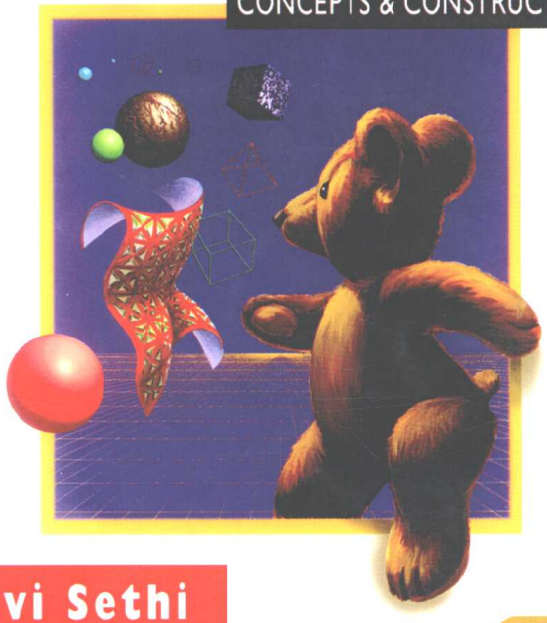
程序设计语言
概念和结构

（英文版 · 第2版）

2nd Edition

Programming Languages

CONCEPTS & CONSTRUCTS



Ravi Sethi

（美）Ravi Sethi 著

机械工业出版社
China Machine Press

Addison-Wesley

程序设计语言
概念和结构

经 典 原 版 书 库

（英文版 · 第2版）

程序设计语言
概念和结构

Programming Languages
Concepts and Concepts
(Second Edition)

(美) Ravi Sethi (贝尔实验室) 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2001-5009

图书在版编目（CIP）数据

程序设计语言：概念和结构（英文版·第2版）/（美）塞西（Sethi, R.）著. -北京：机械工业出版社，2002.1

（经典原版书库）

书名原文：Programming Languages: Concepts and Constructs, 2E

ISBN 7-111-09594-4

Ⅰ.程… Ⅱ.塞… Ⅲ.程序语言-英文 Ⅳ.TP312

中国版本图书馆CIP数据核字（2001）第088567号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：华 章

北京昌平奔腾印刷厂印刷·新华书店北京发行所发行

2002年1月第1版·2002年3月第2次印刷

880mm×1230mm 1/32·20.875印张

印数：2 001-4 000册

定价：39.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：针对本科生的核心课程，剔抉外版菁华而成“国外经典教材”系列；对影印版的教材，则单独开辟出“经典原版书库”；定位在高级教程和专业参考的“计算机科学丛书”还将保持原来的风格，继续出版新的品种。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

“经典原版书库”是响应教育部提出的使用原版国外教材的号召，为国内高校的计算机教学度身订造的。在广泛地征求并听取丛书的“专家指导委员会”的意见后，我们最终选定了这30多种篇幅内容适度、讲解鞭辟入里的教材，其中的大部分已经被M.I.T.、Stanford、U.C. Berkley、C.M.U.等世界名牌大学采用。丛书不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995265
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

Preface

This book is designed for junior/senior level courses on programming languages. A minimal pre-requisite is an introductory programming course. With supplementary readings, the book can also be used for graduate courses.

What's New in this Edition?

Changes on the language scene and feedback from the use of the book have prompted a thorough revision. Instructors liked the emphasis on concepts, but asked that the concepts be illustrated using fewer languages. Meanwhile, Modula-2 has faded, and C++ has taken off as a language for production programming. Candidates for functional languages now include Standard ML, Haskell, and Miranda.

The new edition has 15 chapters, three more than the first edition. The role of the three new chapters is as follows:

- Data types like arrays, records, and pointers have a new chapter.
- Functional programming is introduced using ML in a new chapter.
- Language summaries appear in a final chapter.

Language description and syntax are now treated early, in Chapter 2.

Organization of this Book

The emphasis is on concepts and how they work together, rather than on language features. Related concepts are therefore covered together, to allow meaningful examples and programming exercises along the way. Just enough of a language is introduced, as needed, for the examples and exercises. Language summaries appear in Chapter 15.

Part I: Introduction

Chapter 1 traces the role and development of programming languages. It introduces the programming paradigms in this book. They include imperative, object-oriented, functional, and logic programming.

Syntax description is treated in Chapter 2, so it can be applied in the rest of the book. The examples in the chapter deal with expressions, since methods for describing the syntax of expressions carry over to the rest of a language.

Part II: Imperative Programming

The imperative family is treated in Chapters 3–5. The term "imperative" comes from command or action; the computation model is that of a sequence of actions on an underlying machine.

Chapter 3 deals with control flow. Structured constructs like **while** statements organize the flow of control so that the unit of programming is a structured statement, instead of an individual assignment. Students in a course that emphasizes imperative programming are usually familiar with Pascal, so this chapter goes beyond assignments and structured statements to consider programming with invariants. The examples deal with basic values, like integers, and arrays.

Chapter 4 deals with data in imperative languages. Data representation facilities such as arrays, records, and pointers, have been stable since Pascal and C appeared. The treatment of these facilities anticipates their use to represent objects in Chapters 6 and 7.

Chapter 5 rounds out the discussion of the core of imperative languages, embodied in a language like Pascal or C. Among the topics are the distinction between the source text of a procedure and its activations, parameter passing, scope rules, and storage allocation.

This book illustrates imperative programming using Pascal, where possible. Pascal suffices as a vehicle for Chapters 3–5. C is an alternative.

Part III: Object-Oriented Programming

As programs get larger, the natural unit of programming is a grouping of data and operations. The progression of concepts for such groupings can be described in terms of modules, user-defined types (for example, stacks), and classes (as in object-oriented programming).

Chapter 6 begins with of programming with procedures, modules, and classes. These constructs serve distinct needs and can be used in combination with each other: procedures are needed to implement operations in a module or class; modules can be used to statically partition the source text of a program with classes. Some versions of Pascal support modules; they can be used for the first half of Chapter 6 as well. C++, an extension of C, is introduced in Chapter 6.

The model of computation in Chapter 7 is that of independent objects. The objects interact by sending messages to each other. The first third of the chapter introduces object-oriented programming in general, using a running example that has similar implementations in C++ and Smalltalk. The rest of the chapter has independent coverage of C++ and Smalltalk, so either one can

be used to explore object-oriented programming. Based on feedback from instructors, this edition covers C++ before Smalltalk, inverting the order in the previous edition. Object-oriented programming is illustrated using both C++ and Smalltalk, since the two represent different approaches.

All of the concepts in Chapters 3–7 can be illustrated using C++. Students can be introduced directly to C++, without going through C.

Part IV: Functional Programming

Functional programming is worth studying as a programming style in its own right; as a setting for studying concepts such as types; and as a technique for language description. The emphasis in Chapter 8 is on concepts, in Chapters 9 and 10 on programming style, and in Chapter 13 on language description. The computational model is based on an expression interpreter; an expression consists of a function applied to subexpressions.

The emphasis in Chapter 8 is on concepts. The simplicity of functional languages makes them convenient for introducing concepts such as values, types, names, and functions. The simplicity results from the emphasis on expressions and values, independent of the underlying machine. The chapter treads ground common to functional languages, using ML as the working language.

The fundamental difference between ML and Lisp is that ML is typed; the influence of types permeates the language. Chapter 9 uses ML to illustrate the use of functions and datatypes. As first-class citizens, functions have the same status as any other values in functional programming. This first-class status permits the creation of powerful operations on collections of data.

Functional programming originated with Lisp. Programs and data are both represented by lists in Lisp; the name is a contraction of "List Processor." The uniform use of lists makes Lisp eminently extensible. Chapter 10 explores the use of lists, using the Scheme dialect of Lisp.

See also Chapter 13, which contains an interpreter for a small subset of Scheme, and Chapter 14, which covers the lambda calculus.

Part V: Other Paradigms

Logic programming goes hand in hand with Prolog, in Chapter 11. Logic programming deals with relations rather than functions. Where it fits, programs are concise, consisting of facts and rules. The languages uses the facts and rules to deduce responses to queries.

Concurrent programming is illustrated using Ada, in Chapter 12. An alternative approach would have been to cover concurrent programming after object-oriented programming. Processes can be formed by giving each object its own thread of computation. The present organization puts functional programming before concurrent programming.

Part VI: Language Description

The methods for language description in Chapter 13 are aimed at specialists. The methods range from attributes used for language translation, to logical rules for used type inference, to interpreters used for clarifying subtle language questions.

A language can be described by writing a definitional interpreter for it, so called because its purpose is to define the interpreted language; efficiency is not a concern. McCarthy's & original definitional interpreter for Lisp in Lisp remains important for language description, so language description is illustrated using the Scheme dialect of Lisp. Chapter 13 develops an interpreter for a small subset of Scheme.

The lambda calculus is the intellectual ancestor of functional languages. The small syntax of the lambda calculus has also led to its use as a vehicle for studying languages. Variants of the lambda calculus are introduced in Chapter 14. The chapter progresses from the pure untyped lambda calculus to typed lambda calculi.

Chapter 15 contains brief summaries of the languages in this book.

Acknowledgments From the First Edition

A graduate seminar at Rutgers University gave me both the opportunity and the incentive to collect material on programming languages. I'd like to thank Alex Borgida, Martin Carroll, Fritz Henglein, Naftaly Minsky, Bob Paige, and Barbara Ryder for keeping the seminar lively.

An undergraduate course at Harvard University used an early draft of this book. Written comments by the students in the course were very helpful.

The organization of this book has benefited greatly from the comments and especially the criticism of the then anonymous reviewers contacted by Addison-Wesley. They are Tom Cheatham, Harvard University, John Crenshaw, Western Kentucky University, Paul Hilfinger, University of California, Berkeley, Barry Kurtz, New Mexico State University, Robert Noonan, College of William and Mary, Ron Olsson, University of California, Davis, William Pervin, University of Texas at Dallas, Paul Reynolds, University of Virginia, David Schmidt, Kansas State University, and Laurie Werth, University of Texas at Austin.

For all their technical help, I am grateful to Al Aho, Jon Bentley, Gerard Berry, Eric Cooper, Bruce Duba, Tom Duncan, Rich Drechsler, Peggy Ellis, Charlie Fischer, Dan Friedman, Georges Gonthier, Bob Harper, Mike Harrison, Bruce Hillyer, Brian Kernighan, Kim King, Chandra Kintala, Dave MacQueen, Dianne Maki, Doug McIlroy, John Mitchell, Mike O'Donnell, Dennis Ritchie, Bjarne Stroustrup, Chris Van Wyk, and Carl Woolf.

This book on programming languages was produced with the help of a number of little languages. The diagrams were drawn using Brian Kernighan's Pic language; the grey-tones in the diagrams rely on the work of

Rich Drechsler. The tables were laid out using Mike Lesk's Tbl program. Eqn, Lorinda Cherry and Brian Kernighan's language for typesetting mathematics, handled the pseudo-code as well. The Troff program was originally written by the late Joe Ossanna and is kept vital by Brian Kernighan. Page layout would have suffered without a new Troff macro package and post-processor by Brian Kernighan and Chris Van Wyk. The indexing programs were supplied by Jon Bentley and Brian Kernighan. Cross references were managed using scripts written with the help of Al Aho for managing the text of the "dragon" book.

Finally, I'd like to thank Bell Labs for its support. I have learnt more from my colleagues here than they might suspect. Whenever a question occurred, someone in the building always seemed to have the answer.

Acknowledgments

I really appreciate the comments I have received on the first edition. The experience of instructors and the frank opinions of reviewers have guided the revision.

Debbie Lafferty of Addison-Wesley has been the voice on the phone through the months, coordinating reviews and credits, and generally keeping the project on track. I now know that the reviewers include Bill Appelbe, Michael Barnett, Manuel E. Bermudez, Ray Ford, Aditya P. Mathur, L. A. Oldroyd, and Hamilton Richards — thanks.

For technical help and discussions, I am grateful to Jon Bentley, Lorinda Cherry, Brian Kernighan, Dave MacQueen, Jon Riecke, Bjarne Stroustrup, and Rich Wolf. My colleagues at Bell Labs have been greatly supportive.

A lot has happened while I have been immersed in the Book, including a death, a birth, a move, a fire. Dianne Maki has helped me navigate through it all.

RS

Contents