PROGRAMMING LANGUAGES

SECOND EDITION

Design and Implementation

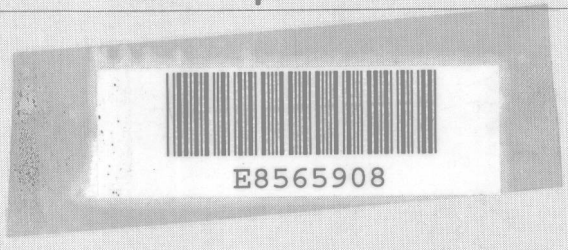
LISP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I ·
RTRAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBO
SP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I · Pa
RAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBOL4
LISP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I ·
RTRAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBO
SP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I · Pa
RAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBOL4
LISP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I ·
RTRAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBO
SP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I · Pa
RAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBOL4
LISP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I ·
RTRAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBO
SP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I · Pa
RAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBOL4
LISP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I ·
RTRAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBO
SP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I · Pa
RAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBOL4
LISP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I ·
RTRAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBO
SP · SNOBOL4 · APL · FORTRAN 77 · COBOL · PL/I · Pa
RAN 77 · COBOL · PL/I · Pascal · Ada · LISP · SNOBOL4

Terrence W. Pratt

second edition

PROGRAMMING LANGUAGES

Design and Implementation



Terrence W. Pratt

Department of Applied Mathematics and Computer Science
University of Virginia

PRENTICE-HALL, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Pratt, Terrence W.
Programming languages.
Bibliography: p. 583
Includes index.
1. Programming languages (Electronic computers)
I. Title.
QA76.7.P7 1984 001.64'24 83-4567
ISBN 0-13-730580-X

Editorial/production supervision by Linda Mihatov
Interior design by Anne Bonanno and Linda Mihatov
Cover design by Anne Bonanno
Manufacturing buyer: Gordon Osbourne

Printed in the United States of America

10 9 8 7 6 5 4

ISBN 0-13-730580-X

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

For Kirsten, Randy, and Laurie

Preface

Computer programming language design and the interplay between language design and implementation are the two central concerns of this book. The design and implementation of programming languages, during the almost thirty-year span from the first version of FORTRAN in the mid-1950s to the design of Ada[1] in the early 1980s, have been more art than science. The underlying principles have always been vague at best, and the accumulation of accepted design alternatives has been far slower than one might expect considering the hundreds of programming languages that have come into existence during the period. The central goal of this book is to bring together the various facets of language design and implementation within a single conceptual framework. The most difficult problem has been to find a framework with both the breadth to encompass the concepts in a wide variety of languages and the depth to allow the relationships among variants of the same concept in different languages to be clearly seen. The result of this endeavor is found in Part I of this book, in which many of the central concepts in programming languages are identified and discussed, along with their implementations on conventional computers. In Part II, eight of the most widely used programming languages are described individually in terms of the concepts developed in Part I. The book is intended as a text for an undergraduate or beginning graduate survey

[1] Ada is a registered trademark of the U.S. Dept. of Defense.

course on programming languages and as a useful reference to concepts, terminology, and languages for practicing programmers.

Part I is organized in the following manner: Chapter 1 develops some of the motivation for the study of programming languages and provides a brief history. Chapter 2 outlines the basic approaches to language implementation. Chapters 3 through 8 form the core of the book, developing the key concepts in the areas of data objects, data types, abstraction mechanisms, control structures, and storage management. *Syntax*, which often makes up a major part of discussions about programming languages, plays a lesser role here. Chapters 3 through 8 are concerned primarily with *semantic structures* and *run-time representations* in languages. Syntax enters only occasionally as a topic of interest, although many examples of various syntactic structures are found in these chapters. Chapter 9 considers directly the topic of syntax and its effect on language structure and translator design. Chapters 10 and 11 round out the conceptual underpinnings through brief discussions of language environments and theoretical models.

Topic selection has been a major problem throughout Part I. What are the central concepts in programming languages? Those selected here have seemed most central, but inevitably some topics have been slighted, and some have been omitted altogether. Doubtless few readers will find the selection entirely to their liking. In an area where there is so little agreement on general principles this cannot be helped. Nevertheless perhaps the overall breadth and balance of the treatment may serve to outweigh somewhat its deficiencies in particular topic areas.

The choice of languages to include in Part II has also been difficult. Two major criteria have guided the selection: widespread use and diversity of concept. On the one hand, inclusion of the most widely used languages has seemed a necessity if the book is to have value as a text and reference. On this basis I have been guided toward the older, more well-established languages: FORTRAN, COBOL, PL/I and Pascal. On the other hand, variety of language design concept is also to be desired if the languages are to exemplify as many of the concepts of Part I as possible. On this basis I have included the list processing language LISP, the string processing language SNOBOL4, and the array processing language APL. Ada is included for its variety of concept, because it promises to be in widespread use within a few years, and because in major aspects it represents the distillation of many language design concepts developed through experience with dozens of other language designs, beginning with ALGOL 60.

Many other languages besides these eight were considered for inclusion, but ultimately restrictions of space and time narrowed the choice. The particular choice of languages should not be too significant, for the intent in Part I is to build a framework for the analysis of languages that may be

applied by the reader to any language. Part II only illustrates how this analysis might be done in eight specific cases.

As a Text

This book is intended as an upper-division undergraduate text for a one-semester course in programming languages (such as the course CS 8 in the ACM Curriculum '78). It is suitable as a beginning graduate text with some supplementary material from the literature. Since it presumes only an elementary background—knowledge of at least one high-level language and a basic knowledge of machine organization and assembly language programming—it could also be used as a lower-division undergraduate text by suitable choice of topics.

A survey course on programming languages is difficult to teach because of the multiplicity of language and implementation concepts that might be treated. There are dozens of languages in fairly wide use, each with its own set of concepts and implementation techniques. How is one to survey this diversity without the result being simply a hodgepodge of unrelated detail? The answer provided by this book is found in the conceptual framework of Part I. The study of a language is organized around the central areas of data objects and data types, abstraction mechanisms, sequence control and data control, storage management, syntax, and operating and programming environments. Part II provides example analyses of eight of the most widely used programming languages, using the concepts developed in Part I. Each chapter provides references for further reading and problems involving, for the most part, application of the concepts to new situations.

There are two different approaches that might be used in a course organized around this material. The simplest approach is to take Part I more or less sequentially, with the instructor providing examples from particular languages to illustrate the material more fully as appropriate. I have found this approach successful with more mature students in a beginning graduate course. At this level the text was supplemented with readings in the literature, and class discussion concentrated on the more complex concepts in the text. The course involved only a few programming problems; the primary emphasis lay with design questions.

The alternative approach, which I have used at the advanced undergraduate level with students having only two or three previous computer science courses, is to work back from the languages to the concepts. The instructor chooses two to four of the languages in Part II, including one that the entering students have already used. The text is supplemented by the usual manuals for these languages, and the students write elementary programs in each language as the course progresses. The appropriate

chapters in Part II, beginning with the chapter on the known language, are taken up after an initial quick pass through Part I. As the concepts in each of the chosen languages are developed and contrasted, the relevant sections of Part I are brought in to provide the necessary depth to support the discussion. By staying close to particular languages, which the student is at the same time applying in practical programming exercises, proper motivation is provided for the study of the general concepts of Part I. The choice of exactly which languages to study in depth is dependent on three factors: the background of the students entering, the languages available on the computer at hand, and the interests of the instructor. For the greatest breadth of concept, at least one of the first five languages in Part II (FORTRAN, COBOL, PL/I, Pascal, Ada) and one of the last three (LISP, SNOBOL4, APL) make an effective combination. It has been my experience that for the undergraduate student a detailed study of a language loses its interest and effectiveness unless coupled with the opportunity to write and run programs in the language at the same time. The organization of the text should allow considerable flexibility in the choice of languages to meet local situations. At the end of the course, if time permits, a particularly useful larger project is to have the student learn and analyze a locally available language that is not described in Part II.

Changes from the First Edition

Our understandings about programming languages have changed in major ways between 1973, when the first edition was being written, and today. As a result, this edition represents almost a complete rewriting of the original text. The central core of the book, Chapters 3 through 8, has been extensively modified. The major changes begin with a new emphasis on the data-object/data-type distinction, and the view of a data type as a set of data objects and the operations on those objects. A more integrated treatment of data types and type checking is used in Chapters 3 and 4 (in contrast to the separate treatment of data and operations in the first edition).

Chapter 5, which is entirely new, brings together a number of concepts treated briefly or not at all in the first edition. The importance of abstraction as a central concept in the construction of programs is emphasized, and the concepts of procedural and data abstraction are introduced. Subprograms are treated for the first time here in their role as abstraction mechanisms. The issues of sequence and data control related to subprograms are taken up in the subsequent chapters.

The discussion of sequence control at the statement level and at the subprogram level (Chapter 6) has been extensively modified and extended. An extended discussion of concurrency is now included, based on the Ada rather than the PL/I approach to tasks and concurrent programming.

Data control structures (scope rules and parameter transmission) are central issues in programming languages. Chapter 7 now deals with these issues in what I hope is a more satisfactory fashion. The distinction between static and dynamic scope is clearly drawn, aliasing is treated, and the entire discussion is reorganized to reflect a better understanding of the issues on my part. In the section on parameter transmission, *call by name* transmission and transmission of label parameters (important topics in ALGOL 60), have been deemphasized.

Other additions include a section on programming environments to complement that on operating environments in Chapter 10 and a brief history of the field in Chapter 1. Deletions have come in two major areas: variable-size data structures (stacks, queues, linked lists) and heap storage management. Several reviewers of the first edition noted that these topics ordinarily are treated in other courses and thus these sections are usually skipped.

Theoretical models (Chapter 11) is a new topic, treated far too superficially. It includes the discussion of universal languages and Turing machines from the first edition, but touches on several additional areas where theory has influenced practice in obvious ways (e.g., compiler construction, program verification, formal semantic definition). It would be more satisfying in many ways to use formal theoretical models throughout the book in a much stronger way, but this is not yet possible. There is no single theoretical framework that encompasses the breadth and depth of topics that need to be treated, and introduction of several different formal structures to treat different topics would obscure rather than clarify. Chapter 11 is intended to provide only motivation for further study of theoretical models and some pointers to the literature.

Finally, the most obvious change: Pascal and Ada have replaced ALGOL 60 in Part II, and Pascal and Ada are the primary sources of examples now in Part I. This change reflects the obvious shift of concerns in language design and implementation during the 1970s.

A Note on Terminology

Many technical terms related to programming languages and language implementations lack a generally accepted definition. Often the same term has been used by different writers to name different concepts, e.g., the term *interpreter*, or alternatively, the same concept has been denoted by a variety of different terms, e.g., the data structure termed a *stack* here, which is also known as a *pushdown list* and *LIFO* (*last-in-first-out*) *list*. I have tried to choose the terminology that seems most generally accepted, e.g., *stack*. In cases where there seems to be little agreement on a standard definition, however, a precise definition is adopted for the purposes of this book (see, for

example, the definition of *interpreter* in Chapter 2), maintaining what seems to be the major denotation of the term insofar as possible. The reader should exercise due caution, however, in the case of terms that are already familiar from other contexts.

A more difficult problem concerns concepts for which there is no generally accepted terminology. In these few cases an appropriate terminology has been introduced in the text. The two most prominent examples are the term *data control structure* (Chapter 7) for those aspects of a language concerned with the visibility and accessibility of data at different points in a program, and the distinction between the operations of *referencing* (Chapter 7) and *selection* (or *accessing*, Chapter 4) on data objects. In choosing these terms, I have tried to avoid conflict with other occasional uses of the same terms, but perhaps without complete success. Again the reader is cautioned.

Terrence W. Pratt
Charlottesville, Virginia

Acknowledgments

This book is based largely on experience teaching programming language concepts at the University of Virginia, and before that at the University of Texas at Austin. A number of people have contributed through their thoughtful reviews of the manuscript or of individual chapters: John Knight, Paul Reynolds, John Gannon, Mary Dee Fosberg, Bob Collins, and Stefan Feyock for the second edition, and Jeffrey Ullman, Ralph Griswold, Saul Rosen, and Dan Friedman for the first. Felix Saltor and Jaume Argila provided an extended review of the first edition, which was extremely helpful. Numerous other users of the first edition also contributed suggestions over the years.

My wife, Barbara Kraft, handled the entry of most of the revised text on our word processor and aided in many other ways in the preparation of the manuscript. The University of Virginia provided financial support, and the Institute for Computer Applications in Science and Engineering (ICASE) at the NASA Langley Research Center provided a congenial working environment for much of the writing. Ruthie Pratt, Ann Patterson, and the University of Texas at Austin contributed to the preparation of the manuscript of the first edition. To these people and institutions, my thanks.

Contents

CHAPTER 2

Programming Language Processors 14

CHAPTER 3

Elementary Data Types 38

CHAPTER 4

Structured Data Types 73