Preface

> A good programming language is a conceptual universe for thinking about programming.
>
> Alan Perlis, NATO Conference on Software Engineering Techniques, Rome, 1969

Programming languages provide the abstractions, organizing principles, and control structures that programmers use to write good programs. This book is about the concepts that appear in programming languages, issues that arise in their implementation, and the way that language design affects program development. The text is divided into four parts:

- *Part 1:* Functions and Foundations
- *Part 2:* Procedures, Types, Memory Management, and Control
- *Part 3:* Modularity, Abstraction, and Object-Oriented Programming
- *Part 4:* Concurrency and Logic Programming

Part 1 contains a short study of Lisp as a worked example of programming language analysis and covers compiler structure, parsing, lambda calculus, and denotational semantics. A short Computability chapter provides information about the limits of compile-time program analysis and optimization.

Part 2 uses procedural Algol family languages and ML to study types, memory management, and control structures.

In Part 3 we look at program organization using abstract data types, modules, and objects. Because object-oriented programming is the most prominent paradigm in current practice, several different object-oriented languages are compared. Separate chapters explore and compare Simula, Smalltalk, C++, and Java.

Part 4 contains chapters on language mechanisms for concurrency and on logic programming.

The book is intended for upper-level undergraduate students and beginning graduate students with some knowledge of basic programming. Students are expected to have some knowledge of C or some other procedural language and some

https://doi.org/10.1017/CBO9780511804175.001 Published online by Cambridge University Press

acquaintance with C++ or some form of object-oriented language. Some experience with Lisp, Scheme, or ML is helpful in Parts 1 and 2, although many students have successfully completed the course based on this book without this background. It is also helpful if students have some experience with simple analysis of algorithms and data structures. For example, in comparing implementations of certain constructs, it will be useful to distinguish between algorithms of constant-, polynomial-, and exponential-time complexity.

After reading this book, students will have a better understanding of the range of programming languages that have been used over the past 40 years, a better understanding of the issues and trade-offs that arise in programming language design, and a better appreciation of the advantages and pitfalls of the programming languages they use. Because different languages present different programming concepts, students will be able to improve their programming by importing ideas from other languages into the programs they write.

Acknowledgments

This book developed as a set of notes for Stanford CS 242, a course in programming languages that I have taught since 1993. Each year, energetic teaching assistants have helped debug example programs for lectures, formulate homework problems, and prepare model solutions. The organization and content of the course have been improved greatly by their suggestions. Special thanks go to Kathleen Fisher, who was a teaching assistant in 1993 and 1994 and taught the course in my absence in 1995. Kathleen helped me organize the material in the early years and, in 1995, transcribed my handwritten notes into online form. Thanks to Amit Patel for his initiative in organizing homework assignments and solutions and to Vitaly Shmatikov for persevering with the glossary of programming language terms. Anne Bracy, Dan Bentley, and Stephen Freund thoughtfully proofread many chapters.

Lauren Cowles, Alan Harvey, and David Tranah of Cambridge University Press were encouraging and helpful. I particularly appreciate Lauren's careful reading and detailed comments of twelve full chapters in draft form. Thanks also are due to the reviewers they enlisted, who made a number of helpful suggestions on early versions of the book. Zena Ariola taught from book drafts at the University of Oregon several years in a row and sent many helpful suggestions; other test instructors also provided helpful feedback.

Finally, special thanks to Krzystof Apt for contributing a chapter on logic programming.

John Mitchell