Preface

Aims and objectives

The aim of this book is to provide a concise introduction to the essentials of programming languages. Although many good books on programming languages exist, most are several hundred pages long. We believe that, in an age in which the pressure on the curriculum, lecturers and students increases with every new development, shorter books are becoming more attractive. In addition, we felt from the start that it should be possible to cover the area of programming languages in about 250 pages by focusing on principles rather than specifics. The book is not intended as an introduction to any existing language in particular.

Today's students can expect to be called upon, in the course of their professional careers, to work with several different programming paradigms, some of which have not yet been invented. To prepare the student for this, the book contains chapters on language principles for each of the four paradigms most in evidence today: imperative, object-oriented, functional and logic. It contains an additional chapter on parallel and distributed programming, since they will become more and more important, and a chapter covering half a dozen less prominent paradigms, in order to widen the student's horizon.

Readers who have finished this book can expect to be able to understand, interpret and evaluate any material, books, articles or manuals on programming languages, and to follow new developments.

Audience

Our audience is students who need a clear view of the field of programming languages but do not need to be experts on any of its aspects. We expect

students to have some experience in programming in one or two imperative languages. Experience of programming in non-imperative languages or of parallel programming is not a prerequisite, and we do not require students to have a mathematical background. We do expect them to be willing to study rather than just read.

The book will also be of interest to professional programmers who have substantial experience in programming, say, C, FORTRAN or Ada. They can use this book for a quick overview of aspects of programming languages they were not aware of, and of different programming paradigms which may become important to them soon (such as object-oriented programming or parallel programming).

We have written this book with a rather heterogeneous audience in mind. In our own course, the audience consists of computer science students in various stages of their education, plus the occasional interested student from a related department. This book is not intended to fit seamlessly into any existing curriculum, but rather to be applicable anywhere an audience of interested, non-novice computer science students can be found.

Use of the book in a course

The size of the book allows it to be used in several ways. With its 270 pages, it can fill most or all of a one-semester course. We have used the book as the main material in a course of 13 weekly lectures of two hours' duration. Experience with this course suggests that the book contains enough material to fill a full 13-week course.

Alternatively, lecturers may prefer to allow some time to discuss additional material on specific languages of their choice, since the book does not cover any language in depth. In our course, for example, we spent the first few lectures on a short introduction to Ada (using a separate book), to acquaint the students with a large real-world language.

Finally, the book can also be used for half-semester courses. These condensed courses are being used at several universities in order to free time for other areas in which students have chosen to become experts. The size of this book makes it more suitable than many others for half-semester courses, although there still might not be enough time to discuss all the chapters in full detail.

Features

To achieve our goal of packing sufficient up-to-date material into a sufficiently short course, this book uses a combination of features: conciseness, extensive coverage of all major paradigms, a simple structure, extensive

treatment of parallel and distributed languages, an overview of other paradigms and chapter summaries.

Conciseness

A key feature of the book is its conciseness, which was achieved not by omitting important issues but by avoiding verbosity and repetitiveness. Difficult concepts are explained in detail and illustrated with examples, but simpler concepts are described briefly. Each feature is explained once rather than in several settings. Also, as said before, we focus on principles rather than specifics. We discuss language-specific features only if we feel they help to explain general principles.

Apart from two or three paragraphs on BNF, the book contains no mathematics and no formalisms. This is a conscious choice. It is clear that a full understanding of the syntax, context conditions and semantics of programming languages can be achieved only through the appropriate formalisms, but such formalisms come at a price. It is our experience that computer science students have to spend considerable amounts of time on a formalism to reach the point at which the knowledge begins to pay off, and we do not expect that kind of time to be available in a course that strives to bring the student up to date in several programming paradigms.

We do not discuss language features that used to intrigue people decades ago but are now outdated, such as computed gotos and call-by-name parameter passing. Short examples are used to illustrate concepts and features, but elaborate case studies have been avoided.

Extensive coverage of all major paradigms

The imperative paradigm is discussed in more detail than the other major paradigms (object-oriented, functional and logic), but the other paradigms are also covered extensively. For each paradigm we discuss the main principles and language concepts, using well-known languages to illustrate them. For each paradigm we also briefly describe other relevant languages.

Simple structure

The book is organized essentially as one chapter per paradigm, which leads to a very simple structure. A more traditional approach would have been to devote several chapters to different language concepts, followed by one chapter for each paradigm, showing how the concepts are applied in the paradigm. Our book does not employ this approach; it describes each concept in the appropriate chapters, where they best make sense. Polymorphism, for example, is discussed in the chapters on object-oriented and functional paradigms, rather than in a chapter of its own.

Extensive treatment of parallel and distributed languages

We think that parallel and distributed systems will play an important role in the near future, so we treat programming languages for such systems in depth, in a separate chapter. We describe recent ideas as well as traditional facilities.

Overview of other paradigms

Many languages outside the major paradigms exist, which students may at first sight just classify as 'funny' languages. Chapter 7 shows how these languages fit in with the rest of the programming world. It shows that the programming language landscape is open-ended, and discusses how to approach less conventional languages such as PIC, APL, SETL and SQL.

Summaries, exercises and bibliographies

Each chapter contains a summary and exercises, with which students can evaluate and test their understanding of the material discussed in the chapter. The answers to most exercises are collected in an appendix. Each chapter also contains a bibliographical section, with references to relevant books and journals. In selecting bibliographical references, we have often preferred more accessible and up-to-date material over original publications. We do not want to put our students off library usage by directing them to references that in the end they find they cannot get hold of. Also, it is easier to obtain references to original papers starting from modern material than the other way round.

Outline of structure

The book contains an introductory chapter, followed by four chapters on each of the major paradigms. Chapter 6 is on parallel and distributed languages and Chapter 7 on other paradigms. Appendix A contains literature references to all languages mentioned in the book, and answers to most exercises can be found in Appendix B. The outline of the seven chapters is given below. Each of Chapters 2 to 6 ends with a section that briefly describes example languages.

Chapter 1 describes *aspects of programming languages* which are relevant for the chapters that follow. It explains how programming languages evolved over the past decades, what exactly a paradigm is and which paradigms are used. It also discusses syntactic and semantic issues. One of the most important tasks of programming languages is to reduce the complexity of programming. Techniques to achieve this (such as decomposition and abstraction) are therefore summarized briefly here. Finally, the

chapter describes how programs are processed (by a compiler or an interpreter), and takes a brief look at programming environments.

Chapter 2 is concerned with the *imperative languages*. It describes the following concepts: data (types and declarations), state (assignment, expressions and input/output), flow of control and program composition. ANSI C and Ada 9X are used to illustrate these concepts.

Chapter 3 addresses *object-oriented languages*. It first discusses the principles of object-oriented design and programming. Many different concepts are used in object-oriented languages, and they interact in subtle ways. Classes and inheritance are introduced, followed by discussions on class hierarchies, types and polymorphism, which all relate to classes. Next, dynamic binding and reference semantics are discussed. These concepts are illustrated by examples written in C++. Some attention is paid to ways of applying the new techniques.

Chapter 4 is on *functional languages*. It first discusses the advantages and disadvantages of functional languages. Next, the concepts of lists, types, polymorphism, higher-order functions, currying, lazy evaluation and equations and pattern matching are discussed and illustrated by Miranda code. Several example applications in Miranda are given.

Chapter 5 describes *logic programming languages*, of which Prolog is by far the most important representative. The main concept in these languages is the Horn clause, which is discussed first. Subsequently, logical variables, relations and data structures are described. Example programs in Prolog are given.

Chapter 6 deals with *parallel and distributed languages*. These languages need ways of expressing parallelism, communication and synchronization. Most of this chapter is devoted to communication and synchronization through shared variables or message passing. The examples are given mainly in a pseudo notation and in Ada 9X.

Chapter 7 shows that many *other paradigms* exist besides those discussed thus far. It describes additional general-purpose paradigms (constraint programming, access-oriented programming and single-datastructure languages) and additional special-purpose paradigms (dataflow programming, little languages, database languages and real-time languages).

Acknowledgements

We owe many thanks to the following people, who were willing to spend the time and effort to read drafts of our book and to supply us with many thoughtful and useful comments: Erik Baalbergen, John Barnes, Raoul Bhoedjang, Leendert van Doorn, Anton Eliëns, Rutger Hofman, Ceriel Jacobs, Frans Kaashoek, Koen Langendoen, Andrew McGettrick, Lourens van der Meij, John Romein, Tim Rühl, Chengzheng Sun, Andy Tanenbaum and the anonymous reviewers.

We also would like to thank the students of the Spring 1993 Pro-

gramming Languages class, Michiel Koens, Paul-Erik Raué and Frans Simon van Leeuwen, for their sometimes very detailed comments.

We are grateful to Addison-Wesley's Simon Plumtree, Alan Grove and Annette Abel for all their help and comments during the writing of this book. We gladly acknowledge the help Ralph Griswold and Wiebren de Jonge provided on detailed questions. Bertrand Meyer helped us with the description of Eiffel and provided the Eiffel code for the example program in Chapter 3.

In the compilation of the bibliography we have benefited greatly from the electronic library system Melvyl, kindly made available to the public by the University of California.

We thank the Faculteit Wiskunde en Informatica of the Vrije Universiteit for the use of the equipment.

Henri E. Bal
Dick Grune

Hoofddorp/Amstelveen
December 1993