Introduction

Facilius per partes in cognitionem totius adducimur
(Seneca, *Epist.* 89,1)

Learning a programming language, for most students in computing, is akin to a rite of passage. It is an important transition, soon recognised as insufficient. Among the tools of the trade, there are many languages, so an important skill for the good computer professional is to know how to move from one language to another (and how to learn new ones) with naturalness and speed.

This competence is not obtained by learning many different languages from scratch. Programming languages, like natural languages, have their similarities, analogies and they inherit characteristics from each other. If it is impossible to learn tens of languages well, it is possible completely to understand the mechanisms that inspire and guide the design and implementation of hundreds of different languages. This knowledge of the "parts" facilitates the understanding of the "whole" of a new language and therefore underpins a fundamental methodological competence in the life of the computing professional, at least as far as it allows them to anticipate innovations and to outlive technologies that grow obsolete.

It is for these reasons that a course on the general aspects of programming languages is, throughout the world, a key step at advanced level for a computing professional (at university or in a profession). The fundamental competences which a computing professional must possess about programming languages are of at least four types:

- Some aspects that are properly considered linguistic.
- Knowledge of how language constructs can be implemented and the relative cost of these implementations.
- Knowledge of those architectural aspects influencing implementation.
- Compilation techniques.

It is rare that a single course deals with all four of these aspects. In particular, description of architectural aspects and compilation techniques are both topics that are sufficiently complex and elaborate to merit independent courses. The remaining 2 aspects are primarily the content of a general course on programming languages and comprise the principle subject of this book.

The literature is rich in texts dealing with these subjects. Generations of students have used them in their learning. All these texts, though, have in mind an advanced reader who already understands many different programming languages, who already has a more than superficial competence with fundamental mechanisms and who is not afraid when confronted by a fragment of code written in an unknown language (because they are able understand it by analogy using the differences between it and what they already know). These are texts, then, that we can say are on "comparative languages". These are long, deep and stimulating, but they are *too* long and deep (read: difficult) for the student who begins their career with a single programming language (or at most 2) and who still has to learn the basic concepts in detail.

This text aims to fill this gap. Experts will see that the content in large measure reflects classical themes. But these very themes are treated in an elementary fashion, assuming only the indispensable minimum of prerequisites. The book also avoids being a catalogue of the differences between different existing programming languages. The ideal (or reference) reader is one who knows one language (well) (for example, Pascal, C, C++ or Java). It is better if they have had some exposure to another language or paradigm. References to languages that are now obsolete have also been avoided and code examples are rarely written in a specific programming language. The text freely uses a sort of pseudo-language (whose concrete syntax was inspired by C and Java) and seeks, in this way, to describe the most relevant aspects of different languages.

Every so often, the boxes at the top of pages contain development of material or a note on a basic concept or something specific about common languages (C, C++, Java; ML and LISP for functional languages; PROLOG for logic-programming languages). The material in boxes can almost always be omitted on a first reading.

Every chapter contains a short sequence of exercises which should be understood as a way of demonstrating an understanding of the material. There are no truly difficult exercises or any that require more than 10 minutes for their solution.

Chapter 3 (Foundations) deals with themes that are not usually encountered in a book on programming languages. It is, however, natural, while discussing static semantics and comparing languages, to ask what are the limits to syntactic analysis of programs and whether what can be done in one language can also be done in another. Rather than send the reader to another text, given the cultural and pragmatic relevance of these questions, we decided to answer these questions directly. In an informal but rigorous manner, in the space of a few pages, we present the undecidability of the halting problem. We also show that all general purpose programming languages express the same class of functions. This helps students who do not always have complete courses on foundations understand the principal results on the limitations on computations.

As well as principles, the text also introduces the three principal *programming paradigms*: object oriented (a theme that is already obligatory in computing), functional and logic programming. The need to write an introductory text is the reason for the exclusion of important themes, such as concurrency and scripting languages, whose mastery represent important skills.

Use of the text The text is first of all a university textbook, even if there is an almost total absence of mathematical and formal prerequisites (this lack makes the book suitable for personal study by the professional who wishes to deepen their knowledge of the mechanisms that lie behind the languages they use). The choice of themes and the presentation style were largely influenced by the experience of teaching the content as part of the degree course in Computer Science in the Faculty of Mathematical, Physical and Natural Sciences at the University of Bologna.

In our experience, a course on programming languages for 6 credits in the second year of a 3-year degree course can cover most of the fundamental aspects covered in the first ten chapters (say 4/5 of them) and, perhaps, including a brief outline of one of the remaining paradigms. With increase in student maturity, the quantity of material that can be presented will clearly increase. In a master's degree course, the material could also be completed by a treatment of compilation.

Acknowledgements Our thanks to Giorgio Levi goes beyond the fact that he had the grace to write the Foreword. Both of us owe to him our first understanding of the mechanisms that underpin programming languages. His teaching appears in this book in a way that is anything but marginal.

Ugo Dal Lago drew the figures using METAPOST, Cinzia Di Giusto, Wilmer Ricciotti, Francesco Spegni and Paolo Tacchella read and commented attentively on the drafts of some chapters. The following people pointed out misprints and errors: Irene Borra, Ferdinanda Camporesi, Marco Comini, Michele Filannino, Matteo Friscini, Stefano Gardenghi, Guido Guizzunti, Giacomo Magisano, Flavio Marchi, Fabrizio Massei, Jacopo Mauro, Maurizio Molle, Mirko Orlandelli, Marco Pedicini, Andrea Rappini, Andrea Regoli, Fabiano Ridolfi, Giovanni Rosignoli, Giampiero Travaglini, Fabrizio Giuseppe Ventola. We gladly acknowledge the support of the Dipartimento di Scienze dell'Informazione of the Università di Bologna towards the English translation.

Maurizio Gabbrielli
Bologna

Simone Martini
Bologna