PREFACE

Introduction

This book has been written to support a practically oriented course in programming language translation for senior undergraduates in computer science. More specifically, it is aimed at students who may be quite competent in the art of imperative programming in C++, C# or Java but whose mathematics may be a little weak; students who require only a solid introduction to the subject, so as to provide them with insight into areas of language design and implementation rather than a deluge of theory which they will probably never use again; students who will enjoy fairly extensive case studies of translators for the sorts of languages with which they are most familiar; students who need to be made aware of compiler writing tools and to come to appreciate and know how to use them.

At the same time, the opportunity has been taken to capitalize on the wave of interest in the use and implementation of C# and Java. In particular, the book aims to provide a useful introduction and summary of the lower-level aspects of programming in the JVM and .NET CLR environments. This will, hopefully, also appeal to a certain class of hobbyist and professional who wishes to know more about how these languages can be exploited.

In addition, the book serves as a reference work on the compiler generator Coco/R. The many case studies that make use of this system will be of interest to readers who wish to learn more about a tool that can be used to aid in the construction of a wide variety of syntax-directed applications, not only compilers.

There are several excellent books already extant in the field of compilation. These range from heavyweight tomes, aimed at skilled systems programmers, to academic treatises that pay a lot of attention to theoretical concepts but far less to how these are put to work in practice. This book attempts to mix theory and practice in a disciplined way, showing the advantages of familiarity with theory, introducing the use of attribute grammars and compiler writing tools, and at the same time giving a highly practical and pragmatic development of translators small enough to understand thoroughly, yet large enough to provide considerable challenge in the many exercises that are suggested. While complete in itself, one of its aims is to make its readers confident enough to be able to tackle the challenges of reading the heavyweight books later on in their careers.

The reader is expected to have a reasonable knowledge of programming in an imperative object-oriented language. The book is practically oriented, and the reader who cannot read and write code will have difficulty following quite a lot of the discussion. However, it is difficult to imagine that students taking courses in compiler construction will not have that sort of background!

Overview

The book starts with a fairly simple overview of the translation process, of the constituent parts of a compiler and of the concepts of porting and bootstrapping compilers. This is followed by a chapter on machine architecture and machine emulation, as later case studies make extensive use of code generation for emulated machines. This is a very common strategy in introductory courses and one which is of increased relevance in the modern era of the JVM and the .NET CLR. Chapter 5 introduces the student to the notions of regular expressions, grammars, BNF and EBNF, and the value of being able to specify languages concisely and accurately.

Three chapters follow on formal syntax theory, parsing and the manual construction of scanners and parsers. The usual classifications of grammars and restrictions on practical grammars are discussed in some detail. The material on parsing is kept to a fairly simple level but with a thorough discussion of the necessary conditions for LL(1) parsing. The parsing method treated in most detail is the method of recursive descent; LR parsing is only briefly discussed.

Chapter 9 is on syntax directed translation, and stresses to the reader the importance and usefulness of being able to start from a context-free grammar, adding attributes and actions that allow for the manual or mechanical construction of a program that will handle the system that it defines. Obvious applications come from the field of translators but applications in other areas, such as simple database design, are also suggested.

The next two chapters give a thorough introduction to the use of Coco/R - a compiler generator for use with L-attributed grammars. In addition to a discussion of Cocol, the specification language for this tool, several in-depth case studies are presented.

The next three chapters discuss the use of Coco/R to construct a recursive descent compiler for Parva, a simple C-like source language. The compiler produces code for the PVM, a simple hypothetical stack-based computer modelled loosely on the JVM and CLR.

The final chapters show how all the techniques discussed previously can be put to use in the construction of compilers for a simple object-oriented subset of C# generating assembler code for the JVM and for the .NET CLR.

The text abounds with suggestions for further exploration and includes references to more advanced texts where these can be followed up. Wherever it seems appropriate the opportunity is taken to make the reader more aware of the strong and weak points in topical imperative languages. Examples are drawn from several languages, such as Pascal, Modula-2, Oberon, C and C++, as well as from C# and Java.

Support material

Recently we have witnessed the spectacular rise in popularity of C# and Java as languages suited not only to the development of large scale, portable, secure and networked systems, but also to teaching the principles of modern programming somewhat more comfortably than can be done with C++. C# has been adopted as the main language used in the present text. As the aim of the text is not to focus on intricate C# programming but compiler construction, the supporting software has been written to be as clear and as simple as possible. Complete source code for all the case studies has been provided in an accompanying *Resource Kit*, both in C# and in Java. A conscious decision was taken to use features of these languages that are essentially identical in both, so that Java programmers should have no difficulty in following the text.

The material provided in the *Resource Kit* includes:

- source code for all the grammars, support classes and case studies discussed in the text, arranged chapter by chapter;
- complete code for the C# and Java versions of the Coco/R compiler generator used in the text and described in detail in Chapter 10, along with the frame files that it needs;
- complete descriptions of the Parva and C#Minor languages that were omitted from the text to save printing costs.
- source code for simple I/O libraries that are also of use in applications besides those described here.

Appendix D gives instructions for unpacking the software provided in the *Resource Kit* and installing it on a computer. In the same appendix are found the addresses of various sites on the Internet where this software (and other freely available compiler construction software) can be found in various formats.

The author also maintains a support website (`http://www.scifac.ru.ac.za/resourcekit`) where additional material, updated versions of the software, errata notices and the like will be posted as they become available, and where also can be found the text of an earlier edition of this book (based on C++) and a host of material used in the delivery of courses based on that edition.

Use as a course text

The book can be used for courses of various lengths. By choosing a selection of topics it could be used on courses as short as 5 - 6 weeks (say 15 - 20 hours of lectures and six lab sessions). It could also be used to support longer and more intensive courses. In our university, selected parts of the material have been successfully used for several years in a course of about 35 - 40 hours of lectures, with strictly controlled and structured related laboratory work, given to students in a pre-Honours year. During that time the course has evolved significantly, from one in which theory and formal specification were very low key to the present stage where students have come to appreciate the use of specification and syntax-directed compiler-writing systems as very powerful and useful tools in their armoury.

It is hoped that instructors can select material from the text so as to suit courses tailored to their own interests, and to their students' capabilities. The core of the theoretical material is to be found in Chapters 1, 2, 5, 6, 7, 8 and

9 and it is suggested that this material should form part of any course based on the book. Restricting the selection of material to those chapters would deny students the very important opportunity to see the material in practice, and at least a partial selection of the material in the practically-oriented chapters should be studied. The development of the small Parva compiler in Chapters 12 - 14 is handled in a way that allows for the later sections of those chapters to be omitted if time is short. A very wide variety of laboratory exercises can be selected from those suggested, providing the students with both a challenge and a feeling of satisfaction when they rise to meet that challenge. Chapter 10 is essentially a reference manual and can be left to students to study for themselves when the need arises. Similarly, Chapter 3 falls into the category of background material. The development of the C#Minor compiler could be omitted from shorter courses or used as supplementary material for readers who wish to take the subject further in their own time.

Limitations

It is, perhaps, worth a slight digression to point out some things which the book does not claim to be and to justify some of the decisions made in the selection of material.

In the first place, while it is hoped that it will serve as a useful foundation for students who are already considerably more advanced, a primary aim has been to make the material as accessible as possible to students with a fairly limited background, to enhance that background and to make them somewhat more critical of it. In some cases a student's background will be only in C# or Java. In spite of claims to the contrary, learning to program really well in these languages within the confines of university courses is not easy, and I have found that many students have a very superficial idea of how they really fit together. After a course such as this, many of the pieces of the language jigsaw fit together rather better.

When introducing the use of compiler writing tools, one might follow the many authors who espouse the classic lex/yacc approach. However, there are now a number of excellent LL(1) based tools and these have the advantage that the code which is produced is close to that which might be hand-crafted; at the same time, recursive descent parsing, besides being fairly intuitive, is powerful enough to handle very usable languages. A decade of experience with Coco/R has convinced me that it is an outstandingly well-crafted tool especially suited to teaching compiler construction in a short time.

That the languages used in case studies and their translators are relative toys cannot be denied. The Parva language, for example, supports only integer variables and simple one-dimensional arrays of these, and the object-oriented C#Minor language does not provide inheritance, polymorphism, exception handling or any of the other more sophisticated features of C# or Java. The text is not intended to be a comprehensive treatise on systems programming in general, just on certain selected topics in that area and so very little is said about native machine code generation and optimization, linkers and loaders, the interaction and relationship with an operating system, and so on. These decisions were all taken deliberately, to keep the material readily understandable and as machine-independent as possible. The systems may be toys, but they are very usable toys! Of course the book is then open to the criticism that many of the more difficult topics in translation (such as native code generation and optimization) are effectively not covered at all, and that the student may be deluded into thinking that these areas do not exist. This is not entirely true - the careful reader will find most of these topics mentioned somewhere, and, in any event, the material on code generation for the JVM and CLR is about as topical as it can be.

Good teachers will always want to put something of their own into a course, regardless of the quality of the prescribed textbook. I have found that a useful (though at times highly dangerous) technique is deliberately not to give the best solutions to a problem in a class discussion, with the optimistic aim that students can be persuaded to discover them for themselves and even gain a sense of achievement in so doing. When applied to a book the technique is particularly dangerous but I have tried to exploit it on several occasions, even though it may give the impression that the author is ignorant!

Another dangerous strategy is to give too much away, especially in a book like this aimed at courses where, so far as I am aware, the traditional approach requires that students make far more of the design decisions for themselves than my approach seems to allow. Many of the books in the field do not show enough of how something is actually done - the bridge between what they give and what the student is required to produce is in excess of what is reasonable for a course which is only part of a general curriculum. I have tried to compensate by suggesting what I hope is a very wide range of searching exercises.

Acknowledgements

I am conscious of my gratitude to many people for their help and inspiration while this book has been developed.

Like many others, I am grateful to Niklaus Wirth whose programming languages and whose writings on the subject of compiler construction and language design refute the modern trend towards ever-increasing complexity in these areas, serving as outstanding models of the way in which progress should be made.

This project could not have been completed without the help of Hanspeter Mössenböck (author of the original Coco/R compiler generators) who willingly gave permission for versions of his software to be distributed with the book. His case studies of compilers built with Coco/R directly influenced my own. He also carefully and critically read the manuscript and made many suggestions for its improvement. I am also deeply indebted to John Gough, long-standing friend, compiler writer and author extraordinaire, for hosting a visit to his Programming Languages and Systems group at the Queensland University of Technology in Brisbane. For ten weeks I was able to immerse myself in getting to terms with the JVM and the CLR platforms under his guidance and that of his colleagues Diane Corney and Wayne Kelly.

The C# software described in this book was developed using the SSCLI "Rotor" toolkit released by Microsoft Corporation in 2002. I am grateful to Microsoft, not only for the use of a fine product and for the opportunity to assess its suitability for a project of this nature, but also for the financial support I have enjoyed as a grant holder under the Rotor scheme and for their hospitality at a Rotor workshop in Cambridge in July 2002.

Closer to home, I have benefited from encouragement and positive criticism from several colleagues, including Shaun Bangay, Peter Clayton, Caro Watkins, George Wells, Peter Wentworth and, particularly, Madeleine Wright, who also acted as a superb proof reader of the initial drafts. My thanks are due to Christopher Cockburn for permission to include his description of tonic solfa (used in Chapter 11). Jody Balarin cheerfully and patiently provided incomparable technical support in the installation and maintenance of my hardware and software. Many hours have been spent using the outstanding development environment provided by Sammy Mitchell's Semware editors.

To Rhodes University I am indebted for financial support, the use of computer facilities and for granting me leave to complete the writing of the book. And, of course, several generations of Rhodes University students have contributed in intangible ways by their reaction to my courses.

But, as always, the greatest debt is owed to my wife Sally and my children David and Helen for their love and support through the many hours when they must have wondered where my priorities lay.

Pat Terry
Rhodes University
Grahamstown

Trademarks

Ada is a trademark of the US Department of Defense.
Apple II is a trademark of Apple Corporation.
Borland C++, Turbo C++, TurboPascal and Delphi are trademarks of Borland International Corporation.
GNU C Compiler is a trademark of the Free Software Foundation.
IBM and IBM PC are trademarks of International Business Machines Corporation.
Intel is a registered trademark of Intel Corporation.
Java is a trademark of Sun MicroSystems.
MIPS is a trademark of MIPS computer systems.
Microsoft, MS and MS-DOS are registered trademarks and Windows is a trademark of Microsoft Corporation.
SPARC is a trademark of Sun Microsystems.
UCSD Pascal and UCSD p-System are trademarks of the Regents of the University of California.
UNIX is a registered trademark of AT&T Bell Laboratories.
Z80 is a trademark of Zilog Corporation.