

# COMPILERS AND COMPILER GENERATORS

**an introduction with C++**

© P.D. Terry, Rhodes University, 1996

e-mail [p.terry@ru.ac.za](mailto:p.terry@ru.ac.za)

The Postscript ® edition of this book was derived from the on-line versions available at <http://www.scifac.ru.ac.za/compilers/>, a WWW site that is occasionally updated, and which contains the latest versions of the various editions of the book, with details of how to download compressed versions of the text and its supporting software and courseware.

The original edition of this book, published originally by International Thomson, is now out of print, but has a home page at <http://cs.ru.ac.za/homes/cspt/compbook.htm>. In preparing the on-line edition, the opportunity was taken to correct the few typographical mistakes that crept into the first printing, and to create a few hyperlinks to where the source files can be found.

Feel free to read and use this book for study or teaching, but please respect my copyright and do not distribute it further without my consent. If you do make use of it I would appreciate hearing from you.

## CONTENTS

### **Preface**

### **Acknowledgements**

### **1 Introduction**

- 1.1 Objectives
- 1.2 Systems programs and translators
- 1.3 The relationship between high-level languages and translators

### **2 Translator classification and structure**

- 2.1 T-diagrams
- 2.2 Classes of translator
- 2.3 Phases in translation
- 2.4 Multi-stage translators
- 2.5 Interpreters, interpretive compilers, and emulators

### **3 Compiler construction and bootstrapping**

- 3.1 Using a high-level host language
- 3.2 Porting a high-level translator

- 3.3 Bootstrapping
- 3.4 Self-compiling compilers
- 3.5 The half bootstrap
- 3.6 Bootstrapping from a portable interpretive compiler
- 3.7 A P-code assembler

## **4 Machine emulation**

- 4.1 Simple machine architecture
- 4.2 Addressing modes
- 4.3 Case study 1 - a single-accumulator machine
- 4.4 Case study 2 - a stack-oriented computer

## **5 Language specification**

- 5.1 Syntax, semantics, and pragmatics
- 5.2 Languages, symbols, alphabets and strings
- 5.3 Regular expressions
- 5.4 Grammars and productions
- 5.5 Classic BNF notation for productions
- 5.6 Simple examples
- 5.7 Phrase structure and lexical structure
- 5.8  $\epsilon$ -productions
- 5.9 Extensions to BNF
- 5.10 Syntax diagrams
- 5.11 Formal treatment of semantics

## **6 Simple assemblers**

- 6.1 A simple ASSEMBLER language
- 6.2 One- and two-pass assemblers, and symbol tables
- 6.3 Towards the construction of an assembler
- 6.4 Two-pass assembly
- 6.5 One-pass assembly

## **7 Advanced assembler features**

- 7.1 Error detection
- 7.2 Simple expressions as addresses
- 7.3 Improved symbol table handling - hash tables
- 7.4 Macro-processing facilities
- 7.5 Conditional assembly
- 7.6 Relocatable code
- 7.7 Further projects

## **8 Grammars and their classification**

- 8.1 Equivalent grammars
- 8.2 Case study - equivalent grammars for describing expressions
- 8.3 Some simple restrictions on grammars

- 8.4 Ambiguous grammars
- 8.5 Context sensitivity
- 8.6 The Chomsky hierarchy
- 8.7 Case study - Clang

## **9 Deterministic top-down parsing**

- 9.1 Deterministic top-down parsing
- 9.2 Restrictions on grammars so as to allow LL(1) parsing
- 9.3 The effect of the LL(1) conditions on language design

## **10 Parser and scanner construction**

- 10.1 Construction of simple recursive descent parsers
- 10.2 Case studies
- 10.3 Syntax error detection and recovery
- 10.4 Construction of simple scanners
- 10.5 Case studies
- 10.6 LR parsing
- 10.7 Automated construction of scanners and parsers

## **11 Syntax-directed translation**

- 11.1 Embedding semantic actions into syntax rules
- 11.2 Attribute grammars
- 11.3 Synthesized and inherited attributes
- 11.4 Classes of attribute grammars
- 11.5 Case study - a small student database

## **12 Using Coco/R - overview**

- 12.1 Installing and running Coco/R
- 12.2 Case study - a simple adding machine
- 12.3 Scanner specification
- 12.4 Parser specification
- 12.5 The driver program

## **13 Using Coco/R - Case studies**

- 13.1 Case study - Understanding C declarations
- 13.2 Case study - Generating one-address code from expressions
- 13.3 Case study - Generating one-address code from an AST
- 13.4 Case study - How do parser generators work?
- 13.5 Project suggestions

## **14 A simple compiler - the front end**

- 14.1 Overall compiler structure
- 14.2 Source handling
- 14.3 Error reporting

- 14.4 Lexical analysis
- 14.5 Syntax analysis
- 14.6 Error handling and constraint analysis
- 14.7 The symbol table handler
- 14.8 Other aspects of symbol table management - further types

## **15 A simple compiler - the back end**

- 15.1 The code generation interface
- 15.2 Code generation for a simple stack machine
- 15.3 Other aspects of code generation

## **16 Simple block structure**

- 16.1 Parameterless procedures
- 16.2 Storage management

## **17 Parameters and functions**

- 17.1 Syntax and semantics
- 17.2 Symbol table support for context sensitive features
- 17.3 Actual parameters and stack frames
- 17.4 Hypothetical stack machine support for parameter passing
- 17.5 Context sensitivity and LL(1) conflict resolution
- 17.6 Semantic analysis and code generation
- 17.7 Language design issues

## **18 Concurrent programming**

- 18.1 Fundamental concepts
- 18.2 Parallel processes, exclusion and synchronization
- 18.3 A semaphore-based system - syntax, semantics, and code generation
- 18.4 Run-time implementation

## **Appendix A: Software resources for this book**

## **Appendix B: Source code for the Clang compiler/interpreter**

## **Appendix C: Cocol grammar for the Clang compiler/interpreter**

## **Appendix D: Source code for a macro assembler**

## **Bibliography**

## **Index**