

Contents

1	Introduction	1
1.1	High-Level Languages	1
1.1.1	Advantages of High-Level Languages	2
1.1.2	Disadvantages of High-Level Languages	3
1.2	High-Level Language Implementation	5
1.2.1	Compilers	6
1.2.2	Compiler Complexity	6
1.2.3	Interpreters	7
1.3	Why Study Compilers?	9
1.4	Present and Future	10
1.5	Conclusions and Further Reading	11
	References	12
2	Compilers and Interpreters	13
2.1	Approaches to Programming Language Implementation	13
2.1.1	Compile or Interpret?	15
2.2	Defining a Programming Language	16
2.2.1	BNF and Variants	17
2.2.2	Semantics	21
2.3	Analysis of Programs	22
2.3.1	Grammars	22
2.3.2	Chomsky Hierarchy	23
2.3.3	Parsing	24
2.4	Compiler and Interpreter Structure	28
2.4.1	Lexical Analysis	29
2.4.2	Syntax Analysis	30
2.4.3	Semantic Analysis	31
2.4.4	Machine-Independent Optimisation	31
2.4.5	Code Generation	32
2.4.6	Machine-Dependent Optimisation	32

2.4.7	Symbol Tables	33
2.4.8	Implementation Issues	33
2.5	Conclusions and Further Reading	34
	References	35
3	Lexical Analysis	37
3.1	Lexical Tokens.	38
3.1.1	An Example	38
3.1.2	Choosing the List of Tokens	39
3.1.3	Issues with Particular Tokens	41
3.1.4	Internal Representation of Tokens	44
3.2	Direct Implementation	45
3.2.1	Planning a Lexical Analyser.	46
3.2.2	Recognising Individual Tokens.	47
3.2.3	More General Issues.	54
3.3	Regular Expressions.	57
3.3.1	Specifying and Using Regular Expressions	57
3.3.2	Recognising Instances of Regular Expressions	58
3.3.3	Finite-State Machines	59
3.4	Tool-Based Implementation	61
3.4.1	Towards a Lexical Analyser for C	62
3.4.2	Comparison with a Direct Implementation	70
3.5	Conclusions and Further Reading	72
	References	73
4	Approaches to Syntax Analysis	75
4.1	Derivations.	75
4.1.1	Leftmost and Rightmost Derivations	76
4.2	Parsing.	77
4.2.1	Top-Down Parsing.	78
4.2.2	Parse Trees and the Leftmost Derivation	78
4.2.3	A Top-Down Parsing Algorithm	82
4.2.4	Classifying Grammars and Parsers	86
4.2.5	Bottom-Up Parsing.	88
4.2.6	Handling Errors	89
4.3	Tree Generation	90
4.4	Conclusions and Further Reading	91
	References	93
5	Practicalities of Syntax Analysis	95
5.1	Top-Down Parsing.	96
5.1.1	A Simple Top-Down Parsing Example.	97
5.1.2	Grammar Transformation for Top-Down Parsing	100

5.2	Bottom-Up Parsing	100
5.2.1	Shift-Reduce Parsers.	101
5.2.2	Bison—A Parser Generator	103
5.3	Tree Generation	110
5.4	Syntax Analysis for DL	113
5.4.1	A Top-Down Syntax Analyser for DL	113
5.4.2	A Bottom-Up Syntax Analyser for DL.	124
5.4.3	Top-Down or Bottom-Up?	131
5.5	Error Handling	132
5.6	Declarations and Symbol Tables	134
5.7	What Can Go Wrong?	136
5.8	Conclusions and Further Reading	137
	References	138
6	Semantic Analysis and Intermediate Code	141
6.1	Types and Type Checking	142
6.1.1	Storing Type Information	142
6.1.2	Type Rules	143
6.2	Storage Management	146
6.2.1	Access to Simple Variables	147
6.2.2	Dealing with Scope	147
6.2.3	Functions	148
6.2.4	Arrays and Other Structures	150
6.3	Syntax-Directed Translation	153
6.3.1	Attribute Grammars	153
6.4	Intermediate Code	154
6.4.1	Linear IRs	155
6.4.2	Graph-Based IRs	158
6.5	Practical Considerations	161
6.5.1	A Three-Address Code IR	162
6.5.2	Translation to the IR	163
6.5.3	An Example	171
6.6	Conclusions and Further Reading	173
	References	175
7	Optimisation	177
7.1	Approaches to Optimisation.	178
7.1.1	Design Principles	178
7.2	Local Optimisation and Basic Blocks	180
7.2.1	Constant Folding and Constant Propagation	181
7.2.2	Common Subexpressions	182
7.2.3	Elimination of Redundant Code	186
7.3	Control and Data Flow	187
7.3.1	Non-local Optimisation.	188

7.3.2	Removing Redundant Variables	190
7.3.3	Loop Optimisation	191
7.4	Parallelism	194
7.4.1	Parallel Execution.	196
7.4.2	Detecting Opportunities for Parallelism	197
7.4.3	Arrays and Parallelism	198
7.5	Conclusions and Further Reading	201
	References	202
8	Code Generation.	205
8.1	Target Machines	205
8.1.1	Real Machines	206
8.1.2	Virtual Machines	209
8.2	Instruction Selection.	210
8.3	Register Allocation	212
8.3.1	Live Ranges	214
8.3.2	Graph Colouring.	215
8.3.3	Complications.	219
8.3.4	Application to DL's Intermediate Representation	219
8.4	Function Call and Stack Management	219
8.4.1	DL Implementation.	220
8.4.2	Call and Return Implementation.	221
8.5	Optimisation.	223
8.5.1	Instruction-Level Parallelism	223
8.5.2	Other Hardware Features	226
8.5.3	Peephole Optimisation	228
8.5.4	Superoptimisation	229
8.6	Automating Code Generator Construction	230
8.7	Conclusions and Further Reading	231
	References	233
9	Implementation Issues	235
9.1	Implementation Strategies	235
9.1.1	Cross-Compilation	237
9.1.2	Implementation Languages	237
9.1.3	Portability.	239
9.2	Additional Software	240
9.3	Particular Requirements	242
9.4	The Future	243
9.5	Conclusions and Further Reading	243
	References	245
	Appendix A: The DL Language	247
	Index	251



<http://www.springer.com/978-3-319-52787-1>

A Practical Approach to Compiler Construction

Watson, D.

2017, XV, 254 p. 26 illus., Softcover

ISBN: 978-3-319-52787-1