

PROGRAMMING LANGUAGE PROCESSORS IN JAVA

COMPILERS AND INTERPRETERS

Suranaree University of Technology



91051000612222

Pearson
Hall

David A Watt & Deryck F Brown

Contents

Preface	xi
1 Introduction	1
1.1 Levels of programming language	1
1.2 Programming language processors	4
1.3 Specification of programming languages	6
1.3.1 Syntax	7
1.3.2 Contextual constraints	15
1.3.3 Semantics	18
1.4 Case study: the programming language Triangle	21
1.5 Further reading	24
Exercises	24
2 Language Processors	26
2.1 Translators and compilers	26
2.2 Interpreters	34
2.3 Real and abstract machines	37
2.4 Interpretive compilers	39
2.5 Portable compilers	40
2.6 Bootstrapping	42
2.6.1 Bootstrapping a portable compiler	43
2.6.2 Full bootstrap	44
2.6.3 Half bootstrap	47
2.6.4 Bootstrapping to improve efficiency	48
2.7 Case study: the Triangle language processor	50
2.8 Further reading	51
Exercises	52
3 Compilation	55
3.1 Phases	55
3.1.1 Syntactic analysis	57
3.1.2 Contextual analysis	59
3.1.3 Code generation	61
3.2 Passes	63

3.2.1	Multi-pass compilation	63
3.2.2	One-pass compilation	64
3.2.3	Compiler design issues	66
3.3	Case study: the Triangle compiler	68
3.4	Further reading	70
	Exercises	71
4	Syntactic Analysis	73
4.1	Subphases of syntactic analysis	73
4.1.1	Tokens	74
4.2	Grammars revisited	77
4.2.1	Regular expressions	77
4.2.2	Extended BNF	79
4.2.3	Grammar transformations	80
4.2.4	Starter sets	82
4.3	Parsing	83
4.3.1	The bottom-up parsing strategy	84
4.3.2	The top-down parsing strategy	87
4.3.3	Recursive-descent parsing	89
4.3.4	Systematic development of a recursive-descent parser	93
4.4	Abstract syntax trees	109
4.4.1	Representation	109
4.4.2	Construction	114
4.5	Scanning	118
4.6	Case study: syntactic analysis in the Triangle compiler	124
4.6.1	Scanning	124
4.6.2	Abstract syntax trees	125
4.6.3	Parsing	125
4.6.4	Error handling	127
4.7	Further reading	128
	Exercises	129
5	Contextual Analysis	136
5.1	Identification	136
5.1.1	Monolithic block structure	137
5.1.2	Flat block structure	139
5.1.3	Nested block structure	142
5.1.4	Attributes	144
5.1.5	Standard environment	148
5.2	Type checking	150
5.3	A contextual analysis algorithm	153
5.3.1	Decoration	153
5.3.2	Visitor classes and objects	154
5.3.3	Contextual analysis as a visitor object	157
5.4	Case study: contextual analysis in the Triangle compiler	163

5.4.1	Identification	163
5.4.2	Type checking	164
5.4.3	Standard environment	166
5.5	Further reading	168
	Exercises	169
6	Run-Time Organization	173
6.1	Data representation	174
6.1.1	Primitive types	176
6.1.2	Records	179
6.1.3	Disjoint unions	181
6.1.4	Static arrays	184
6.1.5	Dynamic arrays	188
6.1.6	Recursive types	190
6.2	Expression evaluation	192
6.3	Static storage allocation	196
6.4	Stack storage allocation	197
6.4.1	Accessing local and global variables	198
6.4.2	Accessing nonlocal variables	202
6.5	Routines	207
6.5.1	Routine protocols	208
6.5.2	Static links	213
6.5.3	Arguments	214
6.5.4	Recursion	217
6.6	Heap storage allocation	219
6.6.1	Heap management	221
6.6.2	Explicit storage deallocation	225
6.6.3	Automatic storage deallocation and garbage collection	228
6.7	Run-time organization for object-oriented languages	230
6.8	Case study: the abstract machine TAM	237
6.9	Further reading	239
	Exercises	240
7	Code Generation	250
7.1	Code selection	251
7.1.1	Code templates	251
7.1.2	Special-case code templates	258
7.2	A code generation algorithm	260
7.2.1	Representation of the object program	260
7.2.2	Systematic development of a code generator	261
7.2.3	Control structures	267
7.3	Constants and variables	269
7.3.1	Constant and variable declarations	270
7.3.2	Static storage allocation	275
7.3.3	Stack storage allocation	281

7.4	Procedures and functions	287
7.4.1	Global procedures and functions	287
7.4.2	Nested procedures and functions	290
7.4.3	Parameters	293
7.5	Case study: code generation in the Triangle compiler	297
7.5.1	Entity descriptions	297
7.5.2	Constants and variables	298
7.6	Further reading	300
	Exercises	301
8	Interpretation	305
8.1	Iterative interpretation	306
8.1.1	Iterative interpretation of machine code	306
8.1.2	Iterative interpretation of command languages	311
8.1.3	Iterative interpretation of simple programming languages	314
8.2	Recursive interpretation	320
8.3	Case study: the TAM interpreter	326
8.4	Further reading	330
	Exercises	331
9	Conclusion	334
9.1	The programming language life cycle	334
9.1.1	Design	335
9.1.2	Specification	336
9.1.3	Prototypes	337
9.1.4	Compilers	338
9.2	Error reporting	339
9.2.1	Compile-time error reporting	339
9.2.2	Run-time error reporting	342
9.3	Efficiency	346
9.3.1	Compile-time efficiency	346
9.3.2	Run-time efficiency	347
9.4	Further reading	352
	Exercises	353
	Projects with the Triangle language processor	354

Appendices

A	Answers to Selected Exercises	359
	Answers 1	359
	Answers 2	360
	Answers 3	363
	Answers 4	363
	Answers 5	369
	Answers 6	372

Answers 7	376
Answers 8	381
Answers 9	385
B Informal Specification of the Programming Language Triangle	387
B.1 Introduction	387
B.2 Commands	387
B.3 Expressions	389
B.4 Value-or-variable names	392
B.5 Declarations	393
B.6 Parameters	394
B.7 Type-denoters	397
B.8 Lexicon	398
B.9 Programs	399
C Description of the Abstract Machine TAM	403
C.1 Storage and registers	403
C.2 Instructions	408
C.3 Routines	408
D Class Diagrams for the Triangle Compiler	413
D.1 Compiler	414
D.2 Abstract syntax trees	415
D.2.1 Commands	416
D.2.2 Expressions	417
D.2.3 Value-or-variable names	418
D.2.4 Declarations	419
D.2.5 Parameters	420
D.2.6 Type-denoters	421
D.2.7 Terminals	421
D.3 Syntactic analyzer	422
D.4 Contextual analyzer	423
D.5 Code generator	424
Bibliography	425
Index	429