

Contents

Preface	xv
Part I: Introduction	1
1 Programming languages	3
1.1 Programming linguistics	3
1.1.1 Concepts and paradigms	3
1.1.2 Syntax, semantics, and pragmatics	5
1.1.3 Language processors	6
1.2 Historical development	6
Summary	10
Further reading	10
Exercises	10
Part II: Basic Concepts	13
2 Values and types	15
2.1 Types	15
2.2 Primitive types	16
2.2.1 Built-in primitive types	16
2.2.2 Defined primitive types	18
2.2.3 Discrete primitive types	19
2.3 Composite types	20
2.3.1 Cartesian products, structures, and records	21
2.3.2 Mappings, arrays, and functions	23
2.3.3 Disjoint unions, discriminated records, and objects	27
2.4 Recursive types	33
2.4.1 Lists	33
2.4.2 Strings	35
2.4.3 Recursive types in general	36
2.5 Type systems	37
2.5.1 Static vs dynamic typing	38
2.5.2 Type equivalence	40
2.5.3 The Type Completeness Principle	42
2.6 Expressions	43
2.6.1 Literals	43
2.6.2 Constructions	44
2.6.3 Function calls	46
2.6.4 Conditional expressions	47
2.6.5 Iterative expressions	48
2.6.6 Constant and variable accesses	49

2.7	Implementation notes	49
2.7.1	Representation of primitive types	49
2.7.2	Representation of Cartesian products	50
2.7.3	Representation of arrays	50
2.7.4	Representation of disjoint unions	51
2.7.5	Representation of recursive types	51
	Summary	52
	Further reading	52
	Exercises	52
3	Variables and storage	57
3.1	Variables and storage	57
3.2	Simple variables	58
3.3	Composite variables	59
3.3.1	Total vs selective update	60
3.3.2	Static vs dynamic vs flexible arrays	61
3.4	Copy semantics vs reference semantics	63
3.5	Lifetime	66
3.5.1	Global and local variables	66
3.5.2	Heap variables	68
3.5.3	Persistent variables	71
3.6	Pointers	73
3.6.1	Pointers and recursive types	74
3.6.2	Dangling pointers	75
3.7	Commands	77
3.7.1	Skips	77
3.7.2	Assignments	77
3.7.3	Proper procedure calls	78
3.7.4	Sequential commands	79
3.7.5	Collateral commands	79
3.7.6	Conditional commands	80
3.7.7	Iterative commands	82
3.8	Expressions with side effects	85
3.8.1	Command expressions	86
3.8.2	Expression-oriented languages	87
3.9	Implementation notes	87
3.9.1	Storage for global and local variables	88
3.9.2	Storage for heap variables	89
3.9.3	Representation of dynamic and flexible arrays	90
	Summary	91
	Further reading	91
	Exercises	92
4	Bindings and scope	95
4.1	Bindings and environments	95
4.2	Scope	97

4.2.1	Block structure	97
4.2.2	Scope and visibility	99
4.2.3	Static vs dynamic scoping	100
4.3	Declarations	102
4.3.1	Type declarations	102
4.3.2	Constant declarations	104
4.3.3	Variable declarations	104
4.3.4	Procedure definitions	105
4.3.5	Collateral declarations	105
4.3.6	Sequential declarations	106
4.3.7	Recursive declarations	107
4.3.8	Scopes of declarations	108
4.4	Blocks	108
4.4.1	Block commands	109
4.4.2	Block expressions	110
4.4.3	The Qualification Principle	110
	Summary	111
	Further reading	112
	Exercises	112
5	Procedural abstraction	115
5.1	Function procedures and proper procedures	115
5.1.1	Function procedures	116
5.1.2	Proper procedures	118
5.1.3	The Abstraction Principle	120
5.2	Parameters and arguments	122
5.2.1	Copy parameter mechanisms	124
5.2.2	Reference parameter mechanisms	125
5.2.3	The Correspondence Principle	128
5.3	Implementation notes	129
5.3.1	Implementation of procedure calls	130
5.3.2	Implementation of parameter mechanisms	130
	Summary	131
	Further reading	131
	Exercises	131
	Part III: Advanced Concepts	133
6	Data abstraction	135
6.1	Program units, packages, and encapsulation	135
6.1.1	Packages	136
6.1.2	Encapsulation	137
6.2	Abstract types	140
6.3	Objects and classes	145
6.3.1	Classes	146
6.3.2	Subclasses and inheritance	151

6.3.3	Abstract classes	157
6.3.4	Single vs multiple inheritance	160
6.3.5	Interfaces	162
6.4	Implementation notes	164
6.4.1	Representation of objects	164
6.4.2	Implementation of method calls	165
	Summary	166
	Further reading	167
	Exercises	167
7	Generic abstraction	171
7.1	Generic units and instantiation	171
7.1.1	Generic packages in ADA	172
7.1.2	Generic classes in C++	174
7.2	Type and class parameters	176
7.2.1	Type parameters in ADA	176
7.2.2	Type parameters in C++	180
7.2.3	Class parameters in JAVA	183
7.3	Implementation notes	186
7.3.1	Implementation of ADA generic units	186
7.3.2	Implementation of C++ generic units	187
7.3.3	Implementation of JAVA generic units	188
	Summary	188
	Further reading	189
	Exercises	189
8	Type systems	191
8.1	Inclusion polymorphism	191
8.1.1	Types and subtypes	191
8.1.2	Classes and subclasses	195
8.2	Parametric polymorphism	198
8.2.1	Polymorphic procedures	198
8.2.2	Parameterized types	200
8.2.3	Type inference	202
8.3	Overloading	204
8.4	Type conversions	207
8.5	Implementation notes	208
8.5.1	Implementation of parametric polymorphism	208
	Summary	210
	Further reading	210
	Exercises	211
9	Control flow	215
9.1	Sequencers	215
9.2	Jumps	216
9.3	Escapes	218

9.4	Exceptions	221
9.5	Implementation notes	226
9.5.1	Implementation of jumps and escapes	226
9.5.2	Implementation of exceptions	227
	Summary	227
	Further reading	228
	Exercises	228
10	Concurrency	231
10.1	Why concurrency?	231
10.2	Programs and processes	233
10.3	Problems with concurrency	234
10.3.1	Nondeterminism	234
10.3.2	Speed dependence	234
10.3.3	Deadlock	236
10.3.4	Starvation	237
10.4	Process interactions	238
10.4.1	Independent processes	238
10.4.2	Competing processes	238
10.4.3	Communicating processes	239
10.5	Concurrency primitives	240
10.5.1	Process creation and control	241
10.5.2	Interrupts	243
10.5.3	Spin locks and wait-free algorithms	243
10.5.4	Events	248
10.5.5	Semaphores	249
10.5.6	Messages	251
10.5.7	Remote procedure calls	252
10.6	Concurrent control abstractions	253
10.6.1	Conditional critical regions	253
10.6.2	Monitors	255
10.6.3	Rendezvous	256
	Summary	258
	Further reading	258
	Exercises	259
	Part IV: Paradigms	263
11	Imperative programming	265
11.1	Key concepts	265
11.2	Pragmatics	266
11.2.1	A simple spellchecker	268
11.3	Case study: C	269
11.3.1	Values and types	269
11.3.2	Variables, storage, and control	272

11.3.3	Bindings and scope	274
11.3.4	Procedural abstraction	274
11.3.5	Independent compilation	275
11.3.6	Preprocessor directives	276
11.3.7	Function library	277
11.3.8	A simple spellchecker	278
11.4	Case study: ADA	281
11.4.1	Values and types	281
11.4.2	Variables, storage, and control	282
11.4.3	Bindings and scope	282
11.4.4	Procedural abstraction	283
11.4.5	Data abstraction	283
11.4.6	Generic abstraction	285
11.4.7	Separate compilation	288
11.4.8	Package library	289
11.4.9	A simple spellchecker	289
	Summary	292
	Further reading	293
	Exercises	293
12	Object-oriented programming	297
12.1	Key concepts	297
12.2	Pragmatics	298
12.3	Case study: C++	299
12.3.1	Values and types	300
12.3.2	Variables, storage, and control	300
12.3.3	Bindings and scope	300
12.3.4	Procedural abstraction	301
12.3.5	Data abstraction	302
12.3.6	Generic abstraction	306
12.3.7	Independent compilation and preprocessor directives	307
12.3.8	Class and template library	307
12.3.9	A simple spellchecker	308
12.4	Case study: JAVA	311
12.4.1	Values and types	312
12.4.2	Variables, storage, and control	313
12.4.3	Bindings and scope	314
12.4.4	Procedural abstraction	314
12.4.5	Data abstraction	315
12.4.6	Generic abstraction	317
12.4.7	Separate compilation and dynamic linking	318
12.4.8	Class library	319
12.4.9	A simple spellchecker	320
12.5	Case study: ADA95	322
12.5.1	Types	322
12.5.2	Data abstraction	325

Summary	328
Further reading	328
Exercises	329
13 Concurrent programming	333
13.1 Key concepts	333
13.2 Pragmatics	334
13.3 Case study: ADA95	336
13.3.1 Process creation and termination	336
13.3.2 Mutual exclusion	338
13.3.3 Admission control	339
13.3.4 Scheduling away deadlock	347
13.4 Case study: JAVA	355
13.4.1 Process creation and termination	356
13.4.2 Mutual exclusion	358
13.4.3 Admission control	359
13.5 Implementation notes	361
Summary	363
Further reading	363
Exercises	363
14 Functional programming	367
14.1 Key concepts	367
14.1.1 Eager vs normal-order vs lazy evaluation	368
14.2 Pragmatics	370
14.3 Case study: HASKELL	370
14.3.1 Values and types	370
14.3.2 Bindings and scope	374
14.3.3 Procedural abstraction	376
14.3.4 Lazy evaluation	379
14.3.5 Data abstraction	381
14.3.6 Generic abstraction	382
14.3.7 Modeling state	384
14.3.8 A simple spellchecker	386
Summary	387
Further reading	388
Exercises	389
15 Logic programming	393
15.1 Key concepts	393
15.2 Pragmatics	396
15.3 Case study: PROLOG	396
15.3.1 Values, variables, and terms	396
15.3.2 Assertions and clauses	398
15.3.3 Relations	398
15.3.4 The closed-world assumption	402
15.3.5 Bindings and scope	403

15.3.6	Control	404
15.3.7	Input/output	406
15.3.8	A simple spellchecker	407
	Summary	409
	Further reading	410
	Exercises	410
16	Scripting	413
16.1	Pragmatics	413
16.2	Key concepts	414
16.2.1	Regular expressions	415
16.3	Case study: PYTHON	417
16.3.1	Values and types	418
16.3.2	Variables, storage, and control	419
16.3.3	Bindings and scope	421
16.3.4	Procedural abstraction	421
16.3.5	Data abstraction	422
16.3.6	Separate compilation	424
16.3.7	Module library	425
	Summary	427
	Further reading	427
	Exercises	427
	Part V: Conclusion	429
17	Language selection	431
17.1	Criteria	431
17.2	Evaluation	433
	Summary	436
	Exercises	436
18	Language design	437
18.1	Selection of concepts	437
18.2	Regularity	438
18.3	Simplicity	438
18.4	Efficiency	441
18.5	Syntax	442
18.6	Language life cycles	444
18.7	The future	445
	Summary	446
	Further reading	446
	Exercises	447
	Bibliography	449
	Glossary	453
	Index	465