

The D Programming Language

Andrei Alexandrescu

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Contents

Foreword by Walter Bright	xv
Foreword by Scott Meyers	xix
Preface	xxiii
Intended Audience	xxiv
Organization of the Book	xxiv
A Brief History	xxv
Acknowledgments	xxvi
1 “D”iving In	1
1.1 Numbers and Expressions	3
1.2 Statements	5
1.3 Function Basics	6
1.4 Arrays and Associative Arrays	7
1.4.1 Building a Vocabulary	7
1.4.2 Array Slicing, Type-Generic Functions, Unit Tests	10
1.4.3 Counting Frequencies, Lambda Functions	12
1.5 Basic Data Structures	14
1.6 Interfaces and Classes	20
1.6.1 More Statistics, Inheritance	23
1.7 Values versus References	25
1.8 Summary	27
2 Basic Types, Expressions	29
2.1 Symbols	30
2.1.1 Special Symbols	31
2.2 Literals	32
2.2.1 Boolean Literals	32
2.2.2 Integral Literals	32
2.2.3 Floating-Point Literals	33
2.2.4 Character Literals	34
2.2.5 String Literals	35

2.2.6	Array and Associative Array Literals	39
2.2.7	Function Literals	40
2.3	Operators	42
2.3.1	Lvalues and Rvalues	42
2.3.2	Implicit Numeric Conversions	42
2.3.3	Typing of Numeric Operators	45
2.3.4	Primary Expressions	46
2.3.5	Postfix Expressions	49
2.3.6	Unary Expressions	51
2.3.7	The Power Expression	54
2.3.8	Multiplicative Expressions	54
2.3.9	Additive Expressions	55
2.3.10	Shift Expressions	55
2.3.11	in Expressions	56
2.3.12	Comparison Operators	56
2.3.13	Bitwise OR, XOR, AND	58
2.3.14	Logical AND	59
2.3.15	Logical OR	59
2.3.16	The Conditional Operator	59
2.3.17	Assignment Operators	60
2.3.18	The Comma Operator	60
2.4	Summary and Quick Reference	61
3	Statements	65
3.1	The Expression Statement	65
3.2	The Compound Statement	66
3.3	The if Statement	67
3.4	The static if Statement	68
3.5	The switch Statement	71
3.6	The final switch Statement	72
3.7	Looping Statements	73
3.7.1	The while Statement	73
3.7.2	The do-while Statement	73
3.7.3	The for Statement	74
3.7.4	The foreach Statement	74
3.7.5	foreach on Arrays	75
3.7.6	The continue and break Statements	78
3.8	The goto Statement	78
3.9	The with Statement	80
3.10	The return Statement	81
3.11	The throw and try Statements	81
3.12	The mixin Statement	82

3.13	The scope Statement	84
3.14	The synchronized Statement	88
3.15	The asm Statement	89
3.16	Summary and Quick Reference	89
4	Arrays, Associative Arrays, and Strings	93
4.1	Dynamic Arrays	93
4.1.1	Length	95
4.1.2	Bounds Checking	95
4.1.3	Slicing	97
4.1.4	Copying	98
4.1.5	Comparing for Equality	100
4.1.6	Concatenating	100
4.1.7	Array-wise Expressions	100
4.1.8	Shrinking	102
4.1.9	Expanding	103
4.1.10	Assigning to .length	106
4.2	Fixed-Size Arrays	107
4.2.1	Length	108
4.2.2	Bounds Checking	108
4.2.3	Slicing	109
4.2.4	Copying and Implicit Conversion	109
4.2.5	Comparing for Equality	110
4.2.6	Concatenating	111
4.2.7	Array-wise Operations	111
4.3	Multidimensional Arrays	111
4.4	Associative Arrays	114
4.4.1	Length	114
4.4.2	Reading and Writing Slots	115
4.4.3	Copying	115
4.4.4	Comparing for Equality	116
4.4.5	Removing Elements	116
4.4.6	Iterating	116
4.4.7	User-Defined Types as Keys	117
4.5	Strings	118
4.5.1	Code Points	118
4.5.2	Encodings	119
4.5.3	Character Types	120
4.5.4	Arrays of Characters + Benefits = Strings	121
4.6	Arrays' Maverick Cousin: The Pointer	124
4.7	Summary and Quick Reference	126

5	Data and Functions. Functional Style	131
5.1	Writing and unittesting a Simple Function	131
5.2	Passing Conventions and Storage Classes	134
5.2.1	ref Parameters and Returns	135
5.2.2	in Parameters	135
5.2.3	out Parameters	136
5.2.4	static Data	137
5.3	Type Parameters	138
5.4	Signature Constraints	140
5.5	Overloading	142
5.5.1	Partial Ordering of Functions	144
5.5.2	Cross-Module Overloading	146
5.6	Higher-Order Functions. Function Literals	148
5.6.1	Function Literals versus Delegate Literals	150
5.7	Nested Functions	150
5.8	Closures	152
5.8.1	OK, This Works. Wait, It Shouldn't. Oh, It Does!	154
5.9	Beyond Arrays. Ranges. Pseudo Members	154
5.9.1	Pseudo Members and the @property Attribute	156
5.9.2	reduce—Just Not <i>ad Absurdum</i>	157
5.10	Variadic Functions	159
5.10.1	Homogeneous Variadic Functions	159
5.10.2	Heterogeneous Variadic Functions	160
5.11	Function Attributes	165
5.11.1	Pure Functions	165
5.11.2	The nothrow Function Attribute	168
5.12	Compile-Time Evaluation	169
6	Classes. Object-Oriented Style	175
6.1	Classes	175
6.2	Object Names Are References	177
6.3	It's an Object's Life	181
6.3.1	Constructors	181
6.3.2	Forwarding Constructors	183
6.3.3	Construction Sequence	184
6.3.4	Destruction and Deallocation	186
6.3.5	Tear-Down Sequence	187
6.3.6	Static Constructors and Destructors	188
6.4	Methods and Inheritance	190
6.4.1	A Terminological Smörgåsbord	191
6.4.2	Inheritance Is Subtyping. Static and Dynamic Type	192
6.4.3	Overriding Is Only Voluntary	193

6.4.4	Calling Overridden Methods	194
6.4.5	Covariant Return Types	195
6.5	Class-Level Encapsulation with static Members	196
6.6	Curbing Extensibility with final Methods	197
6.6.1	final Classes	199
6.7	Encapsulation	199
6.7.1	private	200
6.7.2	package	200
6.7.3	protected	200
6.7.4	public	201
6.7.5	export	201
6.7.6	How Much Encapsulation?	201
6.8	One Root to Rule Them All	203
6.8.1	string toString()	205
6.8.2	size_t toHash()	205
6.8.3	bool opEquals(Object rhs)	205
6.8.4	int opCmp(Object rhs)	209
6.8.5	static Object factory(string className)	210
6.9	Interfaces	212
6.9.1	The Non-Virtual Interface (NVI) Idiom	213
6.9.2	protected Primitives	216
6.9.3	Selective Implementation	217
6.10	Abstract Classes	218
6.11	Nested Classes	222
6.11.1	Classes Nested in Functions	223
6.11.2	static Nested Classes	225
6.11.3	Anonymous Classes	226
6.12	Multiple Inheritance	226
6.13	Multiple Subtyping	230
6.13.1	Overriding Methods in Multiple Subtyping Scenarios	231
6.14	Parameterized Classes and Interfaces	233
6.14.1	Heterogeneous Translation, Again	235
6.15	Summary	237
7	Other User-Defined Types	239
7.1	structs	240
7.1.1	Copy Semantics	241
7.1.2	Passing struct Objects to Functions	242
7.1.3	Life Cycle of a struct Object	243
7.1.4	Static Constructors and Destructors	254
7.1.5	Methods	255
7.1.6	static Members	260

7.1.7	Access Specifiers	261
7.1.8	Nesting structs and classes	261
7.1.9	Nesting structs inside Functions	262
7.1.10	Subtyping with structs. The @disable Attribute	263
7.1.11	Field Layout. Alignment	266
7.2	unions	270
7.3	Enumerated Values	272
7.3.1	Enumerated Types	274
7.3.2	enum Properties	275
7.4	alias	276
7.5	Parameterized Scopes with template	278
7.5.1	Eponymous templates	281
7.6	Injecting Code with mixin templates	282
7.6.1	Symbol Lookup inside a mixin	284
7.7	Summary and Reference	285
8	Type Qualifiers	287
8.1	The immutable Qualifier	288
8.1.1	Transitivity	289
8.2	Composing with immutable	291
8.3	immutable Parameters and Methods	292
8.4	immutable Constructors	293
8.5	Conversions involving immutable	295
8.6	The const Qualifier	297
8.7	Interaction between const and immutable	298
8.8	Propagating a Qualifier from Parameter to Result	299
8.9	Summary	300
9	Error Handling	301
9.1	throwing and catching	301
9.2	Types	302
9.3	finally clauses	306
9.4	nothrow Functions and the Special Nature of Throwable	307
9.5	Collateral Exceptions	307
9.6	Stack Unwinding and Exception-Safe Code	309
9.7	Uncaught Exceptions	312
10	Contract Programming	313
10.1	Contracts	314
10.2	Assertions	316
10.3	Preconditions	317
10.4	Postconditions	319

10.5	Invariants	321
10.6	Skipping Contract Checks. Release Builds	324
10.6.1	enforce Is Not (Quite) assert	325
10.6.2	assert(false)	326
10.7	Contracts: Not for Scrubbing Input	327
10.8	Contracts and Inheritance	329
10.8.1	Inheritance and in Contracts	330
10.8.2	Inheritance and out Contracts	332
10.8.3	Inheritance and invariant Contracts	334
10.9	Contracts in Interfaces	334
11	Scaling Up	337
11.1	Packages and Modules	337
11.1.1	import Declarations	338
11.1.2	Module Searching Roots	340
11.1.3	Name Lookup	341
11.1.4	public import Declarations	344
11.1.5	static import Declarations	345
11.1.6	Selective imports	346
11.1.7	Renaming in imports	347
11.1.8	The module Declaration	348
11.1.9	Module Summaries	349
11.2	Safety	353
11.2.1	Defined and Undefined Behavior	354
11.2.2	The @safe, @trusted, and @system Attributes	355
11.3	Module Constructors and Destructors	356
11.3.1	Execution Order within a Module	357
11.3.2	Execution Order across Modules	358
11.4	Documentation Comments	358
11.5	Interfacing with C and C++	359
11.6	deprecated	359
11.7	version Declarations	360
11.8	debug Declarations	361
11.9	D's Standard Library	361
12	Operator Overloading	365
12.1	Overloading Operators	366
12.2	Overloading Unary Operators	367
12.2.1	Using mixin to Consolidate Operator Definitions	368
12.2.2	Postincrement and Postdecrement	369
12.2.3	Overloading the cast Operator	369
12.2.4	Overloading Ternary Operator Tests and if Tests	370

12.3	Overloading Binary Operators	371
12.3.1	Operator Overloading ²	373
12.3.2	Commutativity	373
12.4	Overloading Comparison Operators	375
12.5	Overloading Assignment Operators	376
12.6	Overloading Indexing Operators	377
12.7	Overloading Slicing Operators	379
12.8	The \$ Operator	379
12.9	Overloading foreach	380
12.9.1	foreach with Iteration Primitives	380
12.9.2	foreach with Internal Iteration	381
12.10	Defining Overloaded Operators in Classes	383
12.11	And Now for Something Completely Different: opDispatch	384
12.11.1	Dynamic Dispatch with opDispatch	386
12.12	Summary and Quick Reference	388
13	Concurrency	391
13.1	Concurrentgate	392
13.2	A Brief History of Data Sharing	394
13.3	Look, Ma, No (Default) Sharing	397
13.4	Starting a Thread	399
13.4.1	immutable Sharing	400
13.5	Exchanging Messages between Threads	401
13.6	Pattern Matching with receive	403
13.6.1	First Match	405
13.6.2	Matching Any Message	405
13.7	File Copying—with a Twist	406
13.8	Thread Termination	407
13.9	Out-of-Band Communication	409
13.10	Mailbox Crowding	410
13.11	The shared Type Qualifier	411
13.11.1	The Plot Thickens: shared Is Transitive	412
13.12	Operations with shared Data and Their Effects	413
13.12.1	Sequential Consistency of shared Data	414
13.13	Lock-Based Synchronization with synchronized classes	414
13.14	Field Typing in synchronized classes	419
13.14.1	Temporary Protection == No Escape	419
13.14.2	Local Protection == Tail Sharing	420
13.14.3	Forcing Identical Mutexes	422
13.14.4	The Unthinkable: casting Away shared	423
13.15	Deadlocks and the synchronized Statement	424
13.16	Lock-Free Coding with shared classes	426

<i>Contents</i>	xiii
13.16.1 shared classes	427
13.16.2 A Couple of Lock-Free Structures	427
13.17 Summary	431
Bibliography	433
Index	439