Table of Contents

Preface xxiii

1. **Getting Started** 1

Your Haskell Environment 1
Getting Started with ghci, the Interpreter 2
Basic Interaction: Using ghci as a Calculator 3
Simple Arithmetic 3
An Arithmetic Quirk: Writing Negative Numbers 4
Boolean Logic, Operators, and Value Comparisons 5
Operator Precedence and Associativity 7
Undefined Values, and Introducing Variables 8
Dealing with Precedence and Associativity Rules 8
Command-Line Editing in ghci 9
Lists 9
Operators on Lists 11
Strings and Characters 11
First Steps with Types 12
A Simple Program 15

2. **Types and Functions** 17

Why Care About Types? 17
Haskell's Type System 18
Strong Types 18
Static Types 19
Type Inference 20
What to Expect from the Type System 20
Some Common Basic Types 21
Function Application 22
Useful Composite Data Types: Lists and Tuples 23
Functions over Lists and Tuples 25
Passing an Expression to a Function 26
Function Types and Purity 27

Haskell Source Files, and Writing Simple Functions 27
Just What Is a Variable, Anyway? 28
Conditional Evaluation 29
Understanding Evaluation by Example 32
Lazy Evaluation 32
A More Involved Example 33
Recursion 34
Ending the Recursion 35
Returning from the Recursion 35
What Have We Learned? 36
Polymorphism in Haskell 36
Reasoning About Polymorphic Functions 38
Further Reading 38
The Type of a Function of More Than One Argument 38
Why the Fuss over Purity? 39
Conclusion 40

3. Defining Types, Streamlining Functions 41
Defining a New Data Type 41
Naming Types and Values 43
Type Synonyms 43
Algebraic Data Types 44
Tuples, Algebraic Data Types, and When to Use Each 45
Analogues to Algebraic Data Types in Other Languages 47
Pattern Matching 50
Construction and Deconstruction 51
Further Adventures 52
Variable Naming in Patterns 53
The Wild Card Pattern 53
Exhaustive Patterns and Wild Cards 54
Record Syntax 55
Parameterized Types 57
Recursive Types 58
Reporting Errors 60
A More Controlled Approach 61
Introducing Local Variables 61
Shadowing 62
The where Clause 63
Local Functions, Global Variables 63
The Offside Rule and Whitespace in an Expression 64
A Note About Tabs Versus Spaces 66
The Offside Rule Is Not Mandatory 66
The case Expression 66

Common Beginner Mistakes with Patterns 67
Incorrectly Matching Against a Variable 67
Incorrectly Trying to Compare for Equality 68
Conditional Evaluation with Guards 68

4. Functional Programming 71
Thinking in Haskell 71
A Simple Command-Line Framework 71
Warming Up: Portably Splitting Lines of Text 72
A Line-Ending Conversion Program 75
Infix Functions 76
Working with Lists 77
Basic List Manipulation 78
Safely and Sanely Working with Crashy Functions 79
Partial and Total Functions 79
More Simple List Manipulations 80
Working with Sublists 81
Searching Lists 82
Working with Several Lists at Once 83
Special String-Handling Functions 84
How to Think About Loops 84
Explicit Recursion 85
Transforming Every Piece of Input 87
Mapping over a List 88
Selecting Pieces of Input 90
Computing One Answer over a Collection 90
The Left Fold 92
Why Use Folds, Maps, and Filters? 93
Folding from the Right 94
Left Folds, Laziness, and Space Leaks 96
Further Reading 99
Anonymous (lambda) Functions 99
Partial Function Application and Currying 100
Sections 102
As-patterns 103
Code Reuse Through Composition 104
Use Your Head Wisely 107
Tips for Writing Readable Code 107
Space Leaks and Strict Evaluation 108
Avoiding Space Leaks with seq 108
Learning to Use seq 109

5. **Writing a Library: Working with JSON Data** **111**
A Whirlwind Tour of JSON 111
Representing JSON Data in Haskell 111
The Anatomy of a Haskell Module 113
Compiling Haskell Source 114
Generating a Haskell Program and Importing Modules 114
Printing JSON Data 115
Type Inference Is a Double-Edged Sword 117
A More General Look at Rendering 118
Developing Haskell Code Without Going Nuts 119
Pretty Printing a String 120
Arrays and Objects, and the Module Header 122
Writing a Module Header 123
Fleshing Out the Pretty-Printing Library 124
 Compact Rendering 127
 True Pretty Printing 128
 Following the Pretty Printer 129
Creating a Package 131
 Writing a Package Description 131
 GHC's Package Manager 133
 Setting Up, Building, and Installing 133
Practical Pointers and Further Reading 134

6. **Using Typeclasses** **135**
The Need for Typeclasses 135
What Are Typeclasses? 136
Declaring Typeclass Instances 139
Important Built-in Typeclasses 139
 Show 139
 Read 141
 Serialization with read and show 143
 Numeric Types 144
 Equality, Ordering, and Comparisons 148
Automatic Derivation 148
Typeclasses at Work: Making JSON Easier to Use 149
 More Helpful Errors 151
 Making an Instance with a Type Synonym 151
Living in an Open World 152
 When Do Overlapping Instances Cause Problems? 153
 Relaxing Some Restrictions on Typeclasses 154
 How Does Show Work for Strings? 155
How to Give a Type a New Identity 155
 Differences Between Data and Newtype Declarations 157

Summary: The Three Ways of Naming Types 158
JSON Typeclasses Without Overlapping Instances 159
The Dreaded Monomorphism Restriction 162
Conclusion 163

7. I/O 165
Classic I/O in Haskell 165
Pure Versus I/O 168
Why Purity Matters 169
Working with Files and Handles 169
More on openFile 171
Closing Handles 172
Seek and Tell 172
Standard Input, Output, and Error 173
Deleting and Renaming Files 174
Temporary Files 174
Extended Example: Functional I/O and Temporary Files 175
Lazy I/O 178
hGetContents 178
readFile and writeFile 180
A Word on Lazy Output 181
interact 181
The IO Monad 183
Actions 183
Sequencing 186
The True Nature of Return 187
Is Haskell Really Imperative? 188
Side Effects with Lazy I/O 188
Buffering 189
Buffering Modes 189
Flushing The Buffer 190
Reading Command-Line Arguments 190
Environment Variables 191

8. Efficient File Processing, Regular Expressions, and Filename Matching 193
Efficient File Processing 193
Binary I/O and Qualified Imports 194
Text I/O 195
Filename Matching 197
Regular Expressions in Haskell 198
The Many Types of Result 198
More About Regular Expressions 200
Mixing and Matching String Types 200

Other Things You Should Know 201
Translating a glob Pattern into a Regular Expression 202
An important Aside: Writing Lazy Functions 205
Making Use of Our Pattern Matcher 206
Handling Errors Through API Design 210
Putting Our Code to Work 211

9. I/O Case Study: A Library for Searching the Filesystem 213
The find Command 213
Starting Simple: Recursively Listing a Directory 213
Revisiting Anonymous and Named Functions 214
Why Provide Both mapM and forM? 215
A Naive Finding Function 215
Predicates: From Poverty to Riches, While Remaining Pure 217
Sizing a File Safely 219
The Acquire-Use-Release Cycle 221
A Domain-Specific Language for Predicates 221
Avoiding Boilerplate with Lifting 223
Gluing Predicates Together 224
Defining and Using New Operators 225
Controlling Traversal 226
Density, Readability, and the Learning Process 228
Another Way of Looking at Traversal 229
Useful Coding Guidelines 232
Common Layout Styles 233

10. Code Case Study: Parsing a Binary Data Format 235
Grayscale Files 235
Parsing a Raw PGM File 236
Getting Rid of Boilerplate Code 238
Implicit State 239
The Identity Parser 240
Record Syntax, Updates, and Pattern Matching 241
A More Interesting Parser 242
Obtaining and Modifying the Parse State 242
Reporting Parse Errors 243
Chaining Parsers Together 243
Introducing Functors 244
Constraints on Type Definitions Are Bad 247
Infix Use of fmap 248
Flexible Instances 248
Thinking More About Functors 249
Writing a Functor Instance for Parse 250

Using Functors for Parsing 251
Rewriting Our PGM Parser 252
Future Directions 254

11. Testing and Quality Assurance 255
QuickCheck: Type-Based Testing 256
  Testing for Properties 257
  Testing Against a Model 259
Testing Case Study: Specifying a Pretty Printer 259
  Generating Test Data 259
  Testing Document Construction 262
  Using Lists as a Model 263
  Putting It All Together 264
Measuring Test Coverage with HPC 265

12. Barcode Recognition 269
A Little Bit About Barcodes 269
  EAN-13 Encoding 270
Introducing Arrays 270
  Arrays and Laziness 273
  Folding over Arrays 273
  Modifying Array Elements 274
Encoding an EAN-13 Barcode 275
Constraints on Our Decoder 275
Divide and Conquer 276
Turning a Color Image into Something Tractable 278
  Parsing a Color Image 278
  Grayscale Conversion 279
  Grayscale to Binary and Type Safety 279
What Have We Done to Our Image? 280
Finding Matching Digits 282
  Run Length Encoding 282
  Scaling Run Lengths, and Finding Approximate Matches 283
  List Comprehensions 284
  Remembering a Match's Parity 285
  Chunking a List 287
  Generating a List of Candidate Digits 287
Life Without Arrays or Hash Tables 288
  A Forest of Solutions 288
  A Brief Introduction to Maps 289
  Further Reading 292
Turning Digit Soup into an Answer 292
  Solving for Check Digits in Parallel 292

Completing the Solution Map with the First Digit 294
Finding the Correct Sequence 295
Working with Row Data 295
Pulling It All Together 296
A Few Comments on Development Style 297

13. **Data Structures** **299**
Association Lists 299
Maps 301
Functions Are Data, Too 303
Extended Example: /etc/passwd 304
Extended Example: Numeric Types 307
First Steps 309
Completed Code 311
Taking Advantage of Functions as Data 317
Turning Difference Lists into a Proper Library 318
Lists, Difference Lists, and Monoids 320
General-Purpose Sequences 322

14. **Monads** **325**
325
Revisiting Earlier Code Examples 325
Maybe Chaining 325
Implicit State 326
Looking for Shared Patterns 327
The Monad Typeclass 329
And Now, a Jargon Moment 330
Using a New Monad: Show Your Work! 331
Information Hiding 331
Controlled Escape 332
Leaving a Trace 332
Using the Logger Monad 333
Mixing Pure and Monadic Code 334
Putting a Few Misconceptions to Rest 336
Building the Logger Monad 336
Sequential Logging, Not Sequential Evaluation 337
The Writer Monad 337
The Maybe Monad 338
Executing the Maybe Monad 338
Maybe at Work, and Good API Design 338
The List Monad 340
Understanding the List Monad 342
Putting the List Monad to Work 343

Desugaring of do Blocks 344
Monads as a Programmable Semicolon 345
Why Go Sugar-Free? 346
The State Monad 346
Almost a State Monad 347
Reading and Modifying the State 348
Will the Real State Monad Please Stand Up? 348
Using the State Monad: Generating Random Values 349
A First Attempt at Purity 350
Random Values in the State Monad 351
Running the State Monad 352
What About a Bit More State? 352
Monads and Functors 354
Another Way of Looking at Monads 354
The Monad Laws and Good Coding Style 355

15. Programming with Monads 359
Golfing Practice: Association Lists 359
Generalized Lifting 360
Looking for Alternatives 362
The Name mplus Does Not Imply Addition 364
Rules for Working with MonadPlus 364
Failing Safely with MonadPlus 364
Adventures in Hiding the Plumbing 365
Supplying Random Numbers 368
Another Round of Golf 369
Separating Interface from Implementation 369
Multiparameter Typeclasses 370
Functional Dependencies 370
Rounding Out Our Module 371
Programming to a Monad's Interface 372
The Reader Monad 373
A Return to Automated Deriving 374
Hiding the IO Monad 375
Using a newtype 376
Designing for Unexpected Uses 377
Using Typeclasses 378
Isolation and Testing 379
The Writer Monad and Lists 380
Arbitrary I/O Revisited 381

16. Using Parsec 383
First Steps with Parsec: Simple CSV Parsing 383

The sepBy and endBy Combinators 386
Choices and Errors 387
Lookahead 389
Error Handling 390
Extended Example: Full CSV Parser 391
Parsec and MonadPlus 393
Parsing a URL-Encoded Query String 393
Supplanting Regular Expressions for Casual Parsing 395
Parsing Without Variables 395
Applicative Functors for Parsing 395
Applicative Parsing by Example 396
Parsing JSON Data 398
Parsing a HTTP Request 401
Backtracking and Its Discontents 402
Parsing Headers 402

17. Interfacing with C: The FFI 405
Foreign Language Bindings: The Basics 406
Be Careful of Side Effects 407
A High-Level Wrapper 408
Regular Expressions for Haskell: A Binding for PCRE 409
Simple Tasks: Using the C Preprocessor 410
Binding Haskell to C with hsc2hs 411
Adding Type Safety to PCRE 411
Binding to Constants 412
Automating the Binding 413
Passing String Data Between Haskell and C 414
Typed Pointers 416
Memory Management: Let the Garbage Collector Do the Work 417
A High-Level Interface: Marshaling Data 418
Marshaling ByteStrings 419
Allocating Local C Data: The Storable Class 419
Putting It All Together 420
Matching on Strings 422
Extracting Information About the Pattern 423
Pattern Matching with Substrings 424
The Real Deal: Compiling and Matching Regular Expressions 426

18. Monad Transformers 429
Motivation: Boilerplate Avoidance 429
A Simple Monad Transformer Example 430
Common Patterns in Monads and Monad Transformers 431
Stacking Multiple Monad Transformers 433

Hiding Our Work 435
Moving Down the Stack 436
When Explicit Lifting Is Necessary 437
Understanding Monad Transformers by Building One 438
Creating a Monad Transformer 439
More Typeclass Instances 440
Replacing the Parse Type with a Monad Stack 440
Transformer Stacking Order Is Important 441
Putting Monads and Monad Transformers into Perspective 443
Interference with Pure Code 443
Overdetermined Ordering 444
Runtime Overhead 444
Unwieldy Interfaces 444
Pulling It All Together 445

19. Error Handling 447
Error Handling with Data Types 447
Use of Maybe 448
Use of Either 452
Exceptions 454
First Steps with Exceptions 454
Laziness and Exception Handling 455
Using handle 456
Selective Handling of Exceptions 456
I/O Exceptions 457
Throwing Exceptions 459
Dynamic Exceptions 459
Error Handling in Monads 462
A Tiny Parsing Framework 463

20. Systems Programming in Haskell 467
Running External Programs 467
Directory and File Information 468
Program Termination 469
Dates and Times 470
ClockTime and CalendarTime 470
File Modification Times 475
Extended Example: Piping 476
Using Pipes for Redirection 477
Better Piping 483
Final Words on Pipes 491

21. Using Databases **493**
Overview of HDBC 493
Installing HDBC and Drivers 494
Connecting to Databases 495
Transactions 495
Simple Queries 496
SqlValue 497
Query Parameters 497
Prepared Statements 498
Reading Results 499
  Reading with Statements 501
  Lazy Reading 501
Database Metadata 502
Error Handling 503

22. Extended Example: Web Client Programming **505**
Basic Types 506
The Database 506
The Parser 510
Downloading 513
Main Program 515

23. GUI Programming with gtk2hs **517**
Installing gtk2hs 517
Overview of the GTK+ Stack 517
User Interface Design with Glade 518
  Glade Concepts 518
Event-Driven Programming 519
Initializing the GUI 520
The Add Podcast Window 524
Long-Running Tasks 525
Using Cabal 528

24. Concurrent and Multicore Programming **531**
Defining Concurrency and Parallelism 531
Concurrent Programming with Threads 532
  Threads Are Nondeterministic 532
  Hiding Latency 532
Simple Communication Between Threads 533
The Main Thread and Waiting for Other Threads 534
  Safely Modifying an MVar 536
  Safe Resource Management: A Good Idea, and Easy Besides 536
  Finding the Status of a Thread 537

Writing Tighter Code 538
Communicating over Channels 539
Useful Things to Know About 539
MVar and Chan Are Nonstrict 539
Chan Is Unbounded 540
Shared-State Concurrency Is Still Hard 540
Deadlock 541
Starvation 541
Is There Any Hope? 542
Using Multiple Cores with GHC 542
Runtime Options 543
Finding the Number of Available Cores from Haskell 543
Choosing the Right Runtime 544
Parallel Programming in Haskell 544
Normal Form and Head Normal Form 545
Sequential Sorting 545
Transforming Our Code into Parallel Code 545
Knowing What to Evaluate in Parallel 546
What Promises Does par Make? 547
Running Our Code and Measuring Performance 547
Tuning for Performance 550
Parallel Strategies and MapReduce 551
Separating Algorithm from Evaluation 552
Separating Algorithm from Strategy 554
Writing a Simple MapReduce Definition 554
MapReduce and Strategies 555
Sizing Work Appropriately 555
Efficiently Finding Line-Aligned Chunks 557
Counting Lines 558
Finding the Most Popular URLs 559
Conclusions 560

25. Profiling and Optimization 561
Profiling Haskell Programs 561
Collecting Runtime Statistics 562
Time Profiling 563
Space Profiling 566
Controlling Evaluation 570
Strictness and Tail Recursion 571
Adding Strictness 572
Understanding Core 575
Advanced Techniques: Fusion 578
Tuning the Generated Assembly 579

Conclusions 580

26. Advanced Library Design: Building a Bloom Filter 581
Introducing the Bloom Filter 581
Use Cases and Package Layout 582
Basic Design 583
Unboxing, Lifting, and Bottom 583
The ST Monad 584
Designing an API for Qualified Import 585
Creating a Mutable Bloom Filter 586
The Immutable API 587
Creating a Friendly Interface 588
Re-Exporting Names for Convenience 589
Hashing Values 589
Turning Two Hashes into Many 593
Implementing the Easy Creation Function 593
Creating a Cabal Package 595
Dealing with Different Build Setups 596
Compilation Options and Interfacing to C 598
Testing with QuickCheck 599
Polymorphic Testing 600
Writing Arbitrary Instances for ByteStrings 601
Are Suggested Sizes Correct? 602
Performance Analysis and Tuning 604
Profile-Driven Performance Tuning 605

27. Sockets and Syslog 611
Basic Networking 611
Communicating with UDP 611
UDP Client Example: syslog 612
UDP Syslog Server 615
Communicating with TCP 616
Handling Multiple TCP Streams 616
TCP Syslog Server 617
TCP Syslog Client 619

28. Software Transactional Memory 623
The Basics 623
Some Simple Examples 624
STM and Safety 626
Retrying a Transaction 626
What Happens When We Retry? 628
Choosing Between Alternatives 628

Using Higher Order Code with Transactions 628
I/O and STM 629
Communication Between Threads 630
A Concurrent Web Link Checker 631
Checking a Link 633
Worker Threads 634
Finding Links 635
Command-Line Parsing 636
Pattern Guards 637
Practical Aspects of STM 638
Getting Comfortable with Giving Up Control 638
Using Invariants 639

A. Installing GHC and Haskell Libraries 641

B. Characters, Strings, and Escaping Rules 649

Index 655