

BRIEF CONTENTS

Foreword	xix
Preface	xxi
Acknowledgments	xxiii
Introduction	xxv
Chapter 1: Getting Started	1
Chapter 2: Programming a Guessing Game	13
Chapter 3: Common Programming Concepts	31
Chapter 4: Understanding Ownership	59
Chapter 5: Using Structs to Structure Related Data	85
Chapter 6: Enums and Pattern Matching	103
Chapter 7: Managing Growing Projects with Packages, Crates, and Modules	119
Chapter 8: Common Collections	141
Chapter 9: Error Handling	161
Chapter 10: Generic Types, Traits, and Lifetimes	181
Chapter 11: Writing Automated Tests	215
Chapter 12: An I/O Project: Building a Command Line Program	243
Chapter 13: Functional Language Features: Iterators and Closures	273
Chapter 14: More About Cargo and Crates.io	295
Chapter 15: Smart Pointers	315
Chapter 16: Fearless Concurrency	353
Chapter 17: Object-Oriented Programming Features	375
Chapter 18: Patterns and Matching	397

Chapter 19: Advanced Features	419
Chapter 20: Final Project: Building a Multithreaded Web Server	459
Appendix A: Keywords	495
Appendix B: Operators and Symbols.	499
Appendix C: Derivable Traits	507
Appendix D: Useful Development Tools	511
Appendix E: Editions	515
Index	517

CONTENTS IN DETAIL

FOREWORD

xix

PREFACE

xxi

ACKNOWLEDGMENTS

xxiii

INTRODUCTION

xxv

Who Rust Is For	xxvi
Teams of Developers	xxvi
Students	xxvi
Companies	xxvi
Open Source Developers	xxvii
People Who Value Speed and Stability	xxvii
Who This Book Is For	xxvii
How to Use This Book	xxvii
Resources and How to Contribute to This Book	xxix

1

GETTING STARTED

1

Installation	1
Installing rustup on Linux or macOS	2
Installing rustup on Windows	3
Troubleshooting	3
Updating and Uninstalling	4
Local Documentation	4
Hello, World!	4
Creating a Project Directory	4
Writing and Running a Rust Program	5
Anatomy of a Rust Program	5
Compiling and Running Are Separate Steps	6
Hello, Cargo!	7
Creating a Project with Cargo	8
Building and Running a Cargo Project	9
Building for Release	11
Cargo as Convention	11
Summary	11

2

PROGRAMMING A GUESSING GAME

13

Setting Up a New Project	14
Processing a Guess	15
Storing Values with Variables	16
Receiving User Input	16

Handling Potential Failure with Result.	17
Printing Values with println! Placeholders	18
Testing the First Part.	19
Generating a Secret Number.	19
Using a Crate to Get More Functionality	19
Generating a Random Number.	22
Comparing the Guess to the Secret Number	23
Allowing Multiple Guesses with Looping	26
Quitting After a Correct Guess	28
Handling Invalid Input	28
Summary	30

3 COMMON PROGRAMMING CONCEPTS

31

Variables and Mutability	32
Constants	33
Shadowing.	34
Data Types	36
Scalar Types.	36
Compound Types	40
Functions.	43
Parameters	44
Statements and Expressions	46
Functions with Return Values	47
Comments.	49
Control Flow	50
if Expressions	50
Repetition with Loops.	54
Summary	58

4 UNDERSTANDING OWNERSHIP

59

What Is Ownership?	59
Ownership Rules.	61
Variable Scope	61
The String Type.	62
Memory and Allocation	63
Ownership and Functions.	68
Return Values and Scope	69
References and Borrowing	71
Mutable References.	73
Dangling References	75
The Rules of References	77
The Slice Type	77
String Slices	79
Other Slices	83
Summary	83

5 USING STRUCTS TO STRUCTURE RELATED DATA 85

Defining and Instantiating Structs	86
Using the Field Init Shorthand	87
Creating Instances from Other Instances with Struct Update Syntax	88
Using Tuple Structs Without Named Fields to Create Different Types	89
Unit-Like Structs Without Any Fields	89
An Example Program Using Structs	91
Refactoring with Tuples	92
Refactoring with Structs: Adding More Meaning	93
Adding Useful Functionality with Derived Traits	94
Method Syntax	97
Defining Methods	97
Methods with More Parameters	99
Associated Functions	101
Multiple impl Blocks	102
Summary	102

6 ENUMS AND PATTERN MATCHING 103

Defining an Enum	103
Enum Values	104
The Option Enum and Its Advantages Over Null Values	108
The match Control Flow Construct	110
Patterns That Bind to Values	112
Matching with Option<T>	113
Matches Are Exhaustive	114
Catch-All Patterns and the _ Placeholder	115
Concise Control Flow with if let	116
Summary	117

7 MANAGING GROWING PROJECTS WITH PACKAGES, CRATES, AND MODULES 119

Packages and Crates	120
Defining Modules to Control Scope and Privacy	123
Paths for Referring to an Item in the Module Tree	125
Exposing Paths with the pub Keyword	127
Starting Relative Paths with super	130
Making Structs and Enums Public	130
Bringing Paths into Scope with the use Keyword	132
Creating Idiomatic use Paths	133
Providing New Names with the as Keyword	135
Re-exporting Names with pub use	135
Using External Packages	136
Using Nested Paths to Clean Up Large use Lists	137
The Glob Operator	138
Separating Modules into Different Files	138
Summary	140

8 COMMON COLLECTIONS 141

Storing Lists of Values with Vectors	142
Creating a New Vector	142
Updating a Vector	142
Reading Elements of Vectors	143
Iterating Over the Values in a Vector	145
Using an Enum to Store Multiple Types	145
Dropping a Vector Drops Its Elements	146
Storing UTF-8 Encoded Text with Strings	147
What Is a String?	147
Creating a New String	147
Updating a String	148
Indexing into Strings	151
Slicing Strings	152
Methods for Iterating Over Strings	153
Strings Are Not So Simple	154
Storing Keys with Associated Values in Hash Maps	154
Creating a New Hash Map	154
Accessing Values in a Hash Map	155
Hash Maps and Ownership	156
Updating a Hash Map	156
Hashing Functions	158
Summary	159

9 ERROR HANDLING 161

Unrecoverable Errors with panic!	162
Recoverable Errors with Result	165
Matching on Different Errors	166
Propagating Errors	169
To panic! or Not to panic!	175
Examples, Prototype Code, and Tests	175
Cases in Which You Have More Information Than the Compiler	176
Guidelines for Error Handling	176
Creating Custom Types for Validation	177
Summary	179

10 GENERIC TYPES, TRAITS, AND LIFETIMES 181

Removing Duplication by Extracting a Function	182
Generic Data Types	184
In Function Definitions	184
In Struct Definitions	187
In Enum Definitions	188
In Method Definitions	189
Performance of Code Using Generics	191
Traits: Defining Shared Behavior	192
Defining a Trait	192
Implementing a Trait on a Type	193
Default Implementations	195

Traits as Parameters	197
Returning Types That Implement Traits	199
Using Trait Bounds to Conditionally Implement Methods	200
Validating References with Lifetimes	201
Preventing Dangling References with Lifetimes	201
The Borrow Checker	202
Generic Lifetimes in Functions	203
Lifetime Annotation Syntax	205
Lifetime Annotations in Function Signatures	205
Thinking in Terms of Lifetimes	208
Lifetime Annotations in Struct Definitions	209
Lifetime Elision	209
Lifetime Annotations in Method Definitions	212
The Static Lifetime	212
Generic Type Parameters, Trait Bounds, and Lifetimes Together	213
Summary	213

11

WRITING AUTOMATED TESTS

215

How to Write Tests	216
The Anatomy of a Test Function	216
Checking Results with the assert! Macro	219
Testing Equality with the assert_eq! and assert_ne! Macros	222
Adding Custom Failure Messages	224
Checking for Panics with should_panic	226
Using Result<T, E> in Tests	230
Controlling How Tests Are Run	230
Running Tests in Parallel or Consecutively	231
Showing Function Output	231
Running a Subset of Tests by Name	233
Ignoring Some Tests Unless Specifically Requested	235
Test Organization	236
Unit Tests	236
Integration Tests	237
Summary	242

12

AN I/O PROJECT: BUILDING A COMMAND LINE PROGRAM

243

Accepting Command Line Arguments	244
Reading the Argument Values	244
Saving the Argument Values in Variables	246
Reading a File	247
Refactoring to Improve Modularity and Error Handling	248
Separation of Concerns for Binary Projects	249
Fixing the Error Handling	253
Extracting Logic from main	256
Splitting Code into a Library Crate	258
Developing the Library's Functionality with Test-Driven Development	259
Writing a Failing Test	260
Writing Code to Pass the Test	262

Working with Environment Variables	265
Writing a Failing Test for the Case-Insensitive Search Function	265
Implementing the search_case_insensitive Function	266
Writing Error Messages to Standard Error Instead of Standard Output	270
Checking Where Errors Are Written	270
Printing Errors to Standard Error	271
Summary	272

13

FUNCTIONAL LANGUAGE FEATURES: ITERATORS AND CLOSURES

273

Closures: Anonymous Functions That Capture Their Environment	274
Capturing the Environment with Closures	274
Closure Type Inference and Annotation	276
Capturing References or Moving Ownership	278
Moving Captured Values Out of Closures and the Fn Traits	280
Processing a Series of Items with Iterators	284
The Iterator Trait and the next Method	285
Methods That Consume the Iterator	286
Methods That Produce Other Iterators	286
Using Closures That Capture Their Environment	287
Improving Our I/O Project	289
Removing a clone Using an Iterator	289
Making Code Clearer with Iterator Adapters	292
Choosing Between Loops and Iterators	292
Comparing Performance: Loops vs. Iterators	293
Summary	294

14

MORE ABOUT CARGO AND CRATES.IO

295

Customizing Builds with Release Profiles	296
Publishing a Crate to Crates.io	297
Making Useful Documentation Comments	297
Exporting a Convenient Public API with pub use	300
Setting Up a Crates.io Account	304
Adding Metadata to a New Crate	304
Publishing to Crates.io	305
Publishing a New Version of an Existing Crate	306
Deprecating Versions from Crates.io with cargo yank	306
Cargo Workspaces	307
Creating a Workspace	307
Creating the Second Package in the Workspace	308
Installing Binaries with cargo install	312
Extending Cargo with Custom Commands	313
Summary	313

15

SMART POINTERS

315

Using Box<T> to Point to Data on the Heap	316
Using Box<T> to Store Data on the Heap	317
Enabling Recursive Types with Boxes	317

Treating Smart Pointers Like Regular References with Deref	321
Following the Pointer to the Value	322
Using Box<T> Like a Reference	323
Defining Our Own Smart Pointer	323
Implementing the Deref Trait	324
Implicit Deref Coercions with Functions and Methods	325
How Deref Coercion Interacts with Mutability	326
Running Code on Cleanup with the Drop Trait	327
Rc<T>, the Reference Counted Smart Pointer	330
Using Rc<T> to Share Data	331
Cloning an Rc<T> Increases the Reference Count	333
RefCell<T> and the Interior Mutability Pattern	334
Enforcing Borrowing Rules at Runtime with RefCell<T>	334
Interior Mutability: A Mutable Borrow to an Immutable Value	335
Allowing Multiple Owners of Mutable Data with Rc<T> and RefCell<T>	342
Reference Cycles Can Leak Memory	343
Creating a Reference Cycle	343
Preventing Reference Cycles Using Weak<T>	346
Summary	351

16 FEARLESS CONCURRENCY 353

Using Threads to Run Code Simultaneously	354
Creating a New Thread with spawn	355
Waiting for All Threads to Finish Using join Handles	356
Using move Closures with Threads	358
Using Message Passing to Transfer Data Between Threads	361
Channels and Ownership Transference	363
Sending Multiple Values and Seeing the Receiver Waiting	364
Creating Multiple Producers by Cloning the Transmitter	365
Shared-State Concurrency	367
Using Mutexes to Allow Access to Data from One Thread at a Time	367
Similarities Between RefCell<T>/Rc<T> and Mutex<T>/Arc<T>	372
Extensible Concurrency with the Send and Sync Traits	373
Allowing Transference of Ownership Between Threads with Send	373
Allowing Access from Multiple Threads with Sync	373
Implementing Send and Sync Manually Is Unsafe	374
Summary	374

17 OBJECT-ORIENTED PROGRAMMING FEATURES 375

Characteristics of Object-Oriented Languages	376
Objects Contain Data and Behavior	376
Encapsulation That Hides Implementation Details	376
Inheritance as a Type System and as Code Sharing	378
Using Trait Objects That Allow for Values of Different Types	379
Defining a Trait for Common Behavior	380
Implementing the Trait	381
Trait Objects Perform Dynamic Dispatch	384
Implementing an Object-Oriented Design Pattern	384
Defining Post and Creating a New Instance in the Draft State	386
Storing the Text of the Post Content	387

Ensuring the Content of a Draft Post Is Empty	387
Requesting a Review Changes the Post's State	388
Adding approve to Change the Behavior of content	389
Trade-offs of the State Pattern	392
Summary	396

18 PATTERNS AND MATCHING 397

All the Places Patterns Can Be Used	398
match Arms	398
Conditional if let Expressions	399
while let Conditional Loops	400
for Loops	400
let Statements	401
Function Parameters	402
Refutability: Whether a Pattern Might Fail to Match	403
Pattern Syntax	405
Matching Literals.	405
Matching Named Variables	405
Multiple Patterns	406
Matching Ranges of Values with ..=.	406
Destructuring to Break Apart Values.	407
Ignoring Values in a Pattern	411
Extra Conditionals with Match Guards.	415
@ Bindings.	417
Summary	418

19 ADVANCED FEATURES 419

Unsafe Rust	420
Unsafe Superpowers	420
Dereferencing a Raw Pointer	421
Calling an Unsafe Function or Method.	423
Accessing or Modifying a Mutable Static Variable	427
Implementing an Unsafe Trait	429
Accessing Fields of a Union	429
When to Use Unsafe Code.	429
Advanced Traits.	430
Associated Types	430
Default Generic Type Parameters and Operator Overloading	431
Disambiguating Between Methods with the Same Name	433
Using Supertraits.	437
Using the Newtype Pattern to Implement External Traits	439
Advanced Types	440
Using the Newtype Pattern for Type Safety and Abstraction	440
Creating Type Synonyms with Type Aliases	440
The Never Type That Never Returns.	443
Dynamically Sized Types and the Sized Trait	444
Advanced Functions and Closures	446
Function Pointers.	446
Returning Closures	448

Macros	449
The Difference Between Macros and Functions	449
Declarative Macros with <code>macro_rules!</code> for General Metaprogramming	449
Procedural Macros for Generating Code from Attributes	451
How to Write a Custom derive Macro	452
Attribute-Like Macros	457
Function-Like Macros	458
Summary	458

20 FINAL PROJECT: BUILDING A MULTITHREADED WEB SERVER 459

Building a Single-Threaded Web Server	460
Listening to the TCP Connection	461
Reading the Request	462
A Closer Look at an HTTP Request	464
Writing a Response	465
Returning Real HTML	466
Validating the Request and Selectively Responding	467
A Touch of Refactoring	469
Turning Our Single-Threaded Server into a Multithreaded Server	471
Simulating a Slow Request	471
Improving Throughput with a Thread Pool	472
Graceful Shutdown and Cleanup	487
Implementing the Drop Trait on ThreadPool	487
Signaling to the Threads to Stop Listening for Jobs	490
Summary	493

A KEYWORDS 495

Keywords Currently in Use	495
Keywords Reserved for Future Use	497
Raw Identifiers	497

B OPERATORS AND SYMBOLS 499

Operators	499
Non-operator Symbols	502

C DERIVABLE TRAITS 507

Debug for Programmer Output	508
PartialEq and Eq for Equality Comparisons	508
PartialOrd and Ord for Ordering Comparisons	509
Clone and Copy for Duplicating Values	509
Hash for Mapping a Value to a Value of Fixed Size	510
Default for Default Values	510

D	
USEFUL DEVELOPMENT TOOLS	511
Automatic Formatting with rustfmt	511
Fix Your Code with rustfix	512
More Lints with Clippy	513
IDE Integration Using rust-analyzer	514

E	
EDITIONS	515

INDEX	517
--------------	------------