Symbols
+ (addition operator), 39, 149–150, 500
& (ampersand), 17, 500
<> (angle brackets), 502–503
  for declaring lifetime parameters, 205
  for specifying type parameters, 142, 186
-> (arrow), 47, 500
* (asterisk), 500
dereference operator, 71, 321–327, 422
glob operator, 138
multiplication operator, 39
@
(at operator), 417–418, 501
: (colon), 501, 503
  for struct fields, 86
  for trait bounds, 197
{} (curly brackets), 505
  for function bodies, 6, 15
  as placeholders in the println! macro, 18
  scope creation, 47, 74
/ (division operator), 39, 501
. (dot), 500
  for method syntax, 97–98
  for struct field access, 86–87
  for tuple element access, 41
:: (double colon), 502–503
" (double quote), 40, 502
- (hyphen)
  for negation, 500
  for subtraction, 39, 500
+ (multiple trait bound syntax), 198, 500
! (never type), 443–444, 502
() (parentheses), 504
  for function parameters, 6, 15
  for tuples, 40–41
? (question mark operator), 171–175, 501
% (remainder operator), 39, 500
; (semicolon), 6, 501
  in the array type, 42
  to end statements, 47
' (single quote), 502
  for characters, 40
  for lifetime parameter names, 205
  for loop labels, 55
[] (square brackets), 500
  for array creation, 41
  in the array type, 42
  for element access, 42, 143–145
_ (underscore), 502
  as a catch-all pattern, 29, 115–116, 411–413
  in filenames, 5
  as a visual separator in integer literals, 37
| (vertical pipe)
  in closure definitions, 276, 502
  in patterns, 406, 501

A
ABI (application binary interface), 427
abort, 162
absolute paths, 126
addition
  of custom types, 431–433
  of number types, 39
  of strings, 149–150
addition operator (+), 39, 149–150, 500
ahead-of-time compiled, 7
ampersand (&), 17, 500
ancestor modules, 127
angle brackets (<>, 502–503
  for declaring lifetime parameters, 205
  for specifying type parameters, 142, 186
API (application programming interface), 4, 300–303
application binary interface, 427
Arc<T> type, 370–373, 482–484
arguments, 44
arms
  in if expressions, 50
  in match expressions, 24, 111–112
array data type, 41–43
  accessing elements of, 42
  invalid element access, 42–43
  iterating over elements of, 57–58
  slices of, 83
arrow (→), 47, 500
as_bytes method, 77
as keyword, 135
assert_eq! macro, 222–224
assert! macro, 219–222
assert_ne! macro, 224
associated function, 16, 101
associated types, 430–431
associative array. See HashMap<K, V> type
asterisk (*), 500
dereference operator, 71,
  321–327, 422
glob operator, 138
multiplication operator, 39
atomically reference counted, 370–373
at operator (@), 417–418, 501
attribute-like procedural macros, 457
automatic dereferencing, 99
automatic referencing, 99

B
back of house, 123
backtrace, 163–165
backward-compatibility guarantees, xxiii
binary crate, 19, 121, 129, 241, 249
binary literal syntax, 37
binary target, 312
blanket implementations, 200–201
blocking, 357, 363, 368
Boolean data type, 39
borrow checker, 202–209
borrowing, 71–77
Box<T> type, 316–327
break keyword, 28, 54
buffer overread, 163
byte literal syntax, 37, 78, 502

C
Cargo, xxvi, 7–11
  commands
    build, 9–10
    check, 10
    clippy, 513
doc, 23, 297–299
fix, 512–513
fmt, 511–512
install, 312–313
login, 304
new, 8–9, 14, 121, 124
publish, 297–306
run, 10, 309
test, 217–218, 230–236,
  298–299, 311
update, 21
yank, 306
  extending with custom commands, 313
  workspaces, 307–312
Cargo.lock, 9–10, 21–22, 309–310
Cargo.toml, 8–9, 19–22, 121
  dependencies section, 19–20
  package section, 304–305
  profile section, 296–297
  updating crate versions in, 21
carriage return, 465
cfg (configuration) attribute, 236–237
channels, 361–366, 480–486, 490–493
character data type, 40
closed channel, 361
checked_* methods, 38
capture module, 125, 127
client, 460
Clippy, 513
copy method
  deep copy creation, 67
  trade-offs of, 251
Clone trait, 509–510
capturing the environment with,
  274–276, 278–280, 287–289
  moving ownership into, 279–280
moving ownership out of, 280, 284
returning, 448
running in threads, 358–361
type inference in, 276–278

`cmd.exe`, 3, 5, 7
`cmp` method, 23–24
coherence, 195
collections, 141–159
colon (:), 501, 503
  for struct fields, 86
  for trait bounds, 197
command line arguments, accepting, 244–247
command line notation, 2
comments, 49–50, 297–299, 477
companies, xxvi
compiler-driven development, 473
compile-time evaluation. See constant
evaluation
compiling
  with cargo, 9–10
  in release mode, 11
  with rustc, 5–7
compound data types, 40–43
concurrency, 353–374
concurrent programming, 353–354
configuration (cfg) attribute, 236–237
connection, 461–462
cons list, 317–321
constant evaluation, 34
constants, 33–34
  vs. static variables, 428
  vs. variables, 33–34
constructor, 329
`*const T`, 421–423, 500
consume, 284–286
consuming adapters, 286
continue keyword, 29, 54
control flow, 50–58
conventions
  Cargo, 11
  for crate root files, 121
  naming
    of constants, 34
    of files, 5
    of function and variable
    names, 44
  `debug!` macro, 95–96
dangling
  pointer, 75
  reference, 75–76, 201–203,
  208–209
data race, 74, 427–428
data types, 36–43
  annotation of, 26, 36
  compound, 40–43
  scalar, 36–40
data types, 36–43
  of static variables, 428
  of type parameters, 185
  for use paths, 133–134

Crate, 9, 120–121
  binary, 121–122, 129
  binary vs. library, 19
  library, 121–122, 129
  license of, 305
  metadata, 304–305
  publishing, 297–306
  root file of, 121–122
  root module of, 124
  updating versions, 21–22
  using as a dependency, 19–22,
  136–137
  yanking, 306
`crate` keyword, 124
`crate root`, 121–122, 124, 138
`crates.io`
  publishing to, 297–306
  removing from, 306
  setting up an account on, 304
CRLF sequence, 465
CTRL-C, 27, 54
curly brackets ({}), 505
  for function bodies, 6, 15
  as placeholders in the `println!`
    macro, 18
  scope creation, 47, 74
  custom derive procedural macros,
    452–457
D
dangling
  pointer, 75
dangling reference, 75–76, 201–203,
  208–209
data race, 74, 427–428
data types, 36–43
  annotation of, 26, 36
  compound, 40–43
  scalar, 36–40
data types, 36–43
  of static variables, 428
  of type parameters, 185
  for use paths, 133–134

debug! macro, 95–96
deadlock, 355, 372–373
Debug trait, 94–96, 224, 508
decimal literal syntax, 37
declarative macros, 449–451
deep copy, 509
Default trait, 510
default type parameters, 431–433
dependencies section in Cargo.toml, 9, 19–20
dependency, 7, 11, 19–22
deref coercion, 150, 325–327
dereference operator, 71, 321–327
DerefMut trait, 326–327
Deref trait, 321–327, 440
derive annotation, 94–96, 452–457, 507–510
description metadata, 305
destructor, 329
destructuring
  of enums, 409–410
  in patterns, 407–411
  of structs, 407–409, 410–411
  of tuples, 40–41, 411
Dickinson, Emily, 247
dictionary. See HashMap<K, V> type
Dijkstra, Edsger W., 215
Display trait, 94, 148, 200–201, 437–439, 508
division operator (/), 39, 501
doc tests, 298–299
documentation
  comments, 297–299, 477
  offline for Rust, 4
tests, 298–299
  viewing a crate’s, 23
  writing, 297, 299
dot (.), 500
  for method syntax, 97–98
  for struct field access, 86–87
  for tuple element access, 41
double colon (::), 502–503
double free error, 66, 329
double quote (“), 40, 502
Doyle, Sir Arthur Conan, 293
drop function, 64, 329–330
Drop trait, 327–330, 487–493
dynamically sized type (DST), 444–446
dynamic dispatch, 384
dyn keyword, 257, 380
E
editions, xxiii, 9, 498, 513, 515–516
else if expression, 52
else keyword, 50
empty type, 443–444, 502
encapsulation, 119, 123, 376–378
entry method, 157–158
Entry type, 157–158
enumerate method, 78, 401
enums, 103–110
  defining, 103–104
  destructuring, 409–410
  initializer function, 447–448
  instantiating, 104–105
  making public, 131–132
  variants of, 104
environment, 274–276, 278–280, 287–289
environment variables, 265–270
eprintln! macro, 271–272
Eq trait, 508–509
error handling, 161–180
executable file, 6–7, 9
executing code, 6–7, 9
exit status code, 255
expect method, 17–18, 26, 169
expressions, 46–47
extern functions, 426–427
F
fearless concurrency, 354
FFI (Foreign Function Interface), 426
field init shorthand, 87–88
fields, 86
files, 247–248
  naming conventions, 5
  organization, 138–140
filtered-out tests, 233–235
Firefox web browser, xxvi
floating-point data types, 38–39
fn keyword, 15
FnMut trait, 280–281, 447, 475
FnOnce trait, 280–283, 447, 475–476
Fn trait, 280, 447, 475
fn type, 446–448
Foreign Function Interface, 426
for keyword
loop, 57–58
patterns in, 400–401
in trait implementations, 194
format! macro, 150
from function
on the From trait, 171
on String, 63, 148
front of house, 123
fully qualified syntax, 433–437, 447
functional programming, 273–274
function-like procedural macros, 458
function pointers, 446–448
functions, 43–49
arguments to, 44
bodies, statements and expressions in, 46–47
extern, 426–427
vs. macros, 449
making public, 128–129
with multiple return values using a tuple, 70
parameters of, 44–46
patterns in, 402
returning early from, 47
with return values, 47–49

G
Gallant, Andrew, 244
Gamma, Erich, 376
garbage collector (GC), 59, 63
generics, 181–192, 213–214
default types for, 431–433
in enum definitions, 188–189
in function definitions, 184–187
in method definitions, 189–191
performance of, 191–192
in struct definitions, 187–188
get method
on HashMap<K, V>, 155
on Vec<T>, 143–145
getter methods, 99, 179
Git, 8, 11
global variables, 427–428
grapheme clusters, 152–154
grep, 243
guard, 367
guessing game, 13–30

H
hash. See HashMap<K, V> type
hasher, 158
hashing function, 158
HashMap<K, V> type, 154–158
every method on, 157–158
get method on, 155
insert method on, 154–157
iterating over, 155–156
new function on, 154–155
and ownership, 156
updating, 156–158
hash table. See HashMap<K, V> type
Hash trait, 510
heap
allocating on, 60, 317
and the stack, 60–61
Hello, World! program, 4–7
Helm, Richard, 376
hexadecimal literal syntax, 37
Hoare, Tony, 108
HTTP (Hypertext Transfer Protocol), 460, 464–466
hyphen (-)
for negation, 500
for subtraction, 39, 500

I
IDE (integrated development environment), xxvi, 4, 514
if keyword, 50–54
if let syntax, 116–117
patterns in, 399–400
ignore attribute, 235–236
immutability. See mutability
impl keyword
for defining associated functions, 101
for defining methods, 97–101
for implementing traits, 194
impl Trait syntax, 197–200
indexing syntax, 143–145
indirection, 320–321
inheritance, 378–379
input lifetimes, 210
input/output (io) library, 15
installation of Rust, 1–4
instance, 16
integer data types, 36–38
numeric operations with, 39
type suffixes of, 37
integer overflow, 38
integrated development environment, xxvi, 4, 514
integration tests, 236–241
interfaces. See traits
interior mutability, 334–340, 343, 372
invalidated variable, 66–67
io (input/output) library, 15
IpAddr type, 104–106
irrefutable patterns, 403–405
i64 type
architecture dependent size of, 37
indexing collection with, 38
iterator adapters, 286–289
iterators, 284–294
adapters for, 286–289
consuming adapters for, 286
creating with iter method, 77–78
enumerate method on, 78
next method on, 285
performance of, 293–294
iter method, 77–78

J
Johnson, Ralph, 376
JoinHandle<T> type, 356–358

K
Kay, Alan, 375
Keep, Daniel, 451
keywords, 32, 495–498

L
Language Server Protocol, 514
last in, first out ordering, 60
lazy evaluation, 284, 287
len method, 78
let keyword, 16
using patterns with, 401–402
library crate, 7, 19, 121, 129
license identifier value, 305
license metadata, 305
lifetimes, 201–214
annotation of, 203–209
elision, 209–212
line feed, 465
linker, 2
lints, 513
Linux Foundation, 305
Linux installation of Rust, 2
“The Little Book of Rust Macros,” 451
lock, 367–370
loop keyword, 26–28, 54–56
loop labels, 55–56

M
macOS installation of Rust, 2
macro_export annotation, 450
macro_rules! macros, 449–451
macros, 449–458
declarative, 449–451
vs. functions, 449
procedural, 451
main function, 6, 174–175
mangling, 427
map. See HashMap<K, V> type
match expression, 110–116
exhaustiveness of, 114
handling comparison results with, 24
handling error values with, 166–167
handling Result values with, 28–29, 166
patterns in, 398–399
match guard, 415–417
memory leak, 343, 350–351
message passing, 361–366
methods
defined on enums, 107
defined on structs, 97–102
disambiguating, 433–437
getters, 99
method syntax, 97–98
minigrep project, 243–272
mock object, 336–340
mod keyword, 124
modules, 121–125
cheat sheet, 121–123
file paths for, 122
moving to other files, 138–140
module system, 120
module tree, 124–125
monomorphization, 191–192
move keyword, 279–280, 358–361
moving ownership, 64–67
vs. borrowing, 71–72
with function calls, 68–69
with function return values, 69–70
multiple producer, single consumer (mpsc), 362, 365–366
multiple trait bound syntax (+), 198, 500
multiplication, 39
mutability
of references, 73–75
of variables, 32–33
Mutex<T> type, 367–373, 482–484, 485–487
mut keyword
making a reference mutable with, 73–75
making a variable mutable with, 33
vs. shadowing, 35–36
*mut T, 421–423, 500
mutual exclusion, 367

N
namespace, 63, 101, 104
never type (!), 443–444, 502
new function
on HashMap<K, V>, 154–155
on String, 147–148
on Vec<T>, 142
new project setup, using Cargo, 14
newtype pattern, 439–440
null, 108–110
numeric operations, 39

O
object. See HashMap<K, V> type
object-oriented programming (OOP), 375–396
octal literal syntax, 37
1:1 threading model, 355
open source developers, xxvii
operator overloading, 431–433
operators, 499–501
optimizations, 11
Option<T> enum, 108–110, 113–114
Ordering type, 24
Ord trait, 509
orphan rule, 195, 439
output lifetimes, 210
overflowing * methods, 38
overflow of integers, 38
ownership, 59–83
and functions, 68–70
rules, 61
of struct data, 90–91

P
package, 121
package registry, 297–306
package section in Cargo.toml, 304–305
panicking, 38
panic! macro, 162–165, 226–229
vs. Result, 175–180
parallel programming, 353–354
parameters, 44
patterns in, 402
parentheses (()), 504
for function parameters, 6, 15
for tuples, 40–41
parent modules, 125, 127
parse method, 26
PartialEq trait, 224, 508–509
PartialOrd trait, 187, 509
paths, 125–130
absolute, 126
nested, 137
relative, 125–126
%PATH% system variable, 3, 312
patterns, 397–418
binding to values with, 112–113
destructuring in, 407–411
in for loops, 400–401
in function parameters, 402
in if let syntax, 116–117, 399–400
ignoring values in, 411–415
in let statements, 401–402
in match expressions, 110–116, 398–399
refutable vs. irrefutable, 403–405
in while let loops, 400
.pdb file extension, 7
pointer, 60, 315
  dangling, 75
  to data on the heap, 60–61
  raw, 421–423
  smart, 315–351
poisoned mutex, 485
polymorphism, 378–379
PowerShell, 3–4, 6–7, 269–270
prelude, 15, 138
println! macro, 6, 18–19
privacy, 123, 127–129
procedural macros, 451
attribute-like, 457
custom derive, 452–457
function-like, 458
process, 354
proc_macro crate, 452, 454
profiles, 296–297
profile section in Cargo.toml, 296–297
propagating errors, 169–175
pub keyword, 122, 127–129
public, 127–129
  API, 129, 300–303
  making items, 128
  making structs and enums, 130
pub use, 135–136, 300–303
push method, 142
push_str method, 63, 149
Q
question mark operator (?), 171–175, 501
quote crate, 454–456
R
race conditions, 74, 355
RAII (Resource Acquisition Is Initialization), 64
rand crate, 19–23
random number functionality, 19, 22–23
range syntax, 406–407
Range type, 58
raw identifiers, 497–498
raw pointers, 421–423
Rc<T> type, 330–334, 342–351
read_line method, 17–18
receiver, 361–366
recoverable errors, 161–162, 165–175
recursive type, 317–321
re-export, 135–136, 300–303
RefCell<T> type, 334–351
reference counting, 315, 330–334, 370–373
reference cycles, 343–351
references
  for accessing data from multiple places, 17
  and borrowing, 71–77
  dangling, 75–76
dereferencing, 71
mutability of, 73–75
rules of, 77
refutable patterns, 403–405
registry, 20, 297–306
relative path, 125–126, 130
release mode, 11, 38
release profiles, 296–297
remainder operator (%), 39, 500
request line, 464–465
request-response protocol, 460
Resource Acquisition Is Initialization, 64
Result<T, E> type, 17–18, 165–175
  expect method on, 17–18, 169
  vs. panic!, 175–180
type aliases for, 442–443
  unwrap method on, 168
  unwrap_or_else method on, 168
return keyword, 47
return values
  of functions, 47–49
  of loops, 55
  multiple using a tuple, 70
rev method, 58
ripgrep, 244, 312–313
RLS (Rust Language Server), xxvi
.rs file extension, 5
running code, 5–7, 9–10
Rustaceans, 3
rust-analyzer, 514
rustc, 3, 5–7
rustfix, 512–513
rustfmt, xxvi, 6, 511–512
Rust Language Server, xxvi
“The Rustonomicon,” 145, 351, 374
rustup commands, 1–4
  doc, 4
  uninstall, 4
  update, 4
saturating_* methods, 38
scalar data types, 36–40
scope, 62, 120
SCREAMING_SNAKE_CASE, 428
Self keyword, 98
self module, 125
self parameter, 97
Semantic Versioning (SemVer), 19–20, 306
semicolon (;), 6, 501
  in the array type, 42
  to end statements, 47
Send trait, 373–374, 429, 476
sequence, 58
server, 460
shadowing, 34–36
  vs. mut keyword, 35–36
shared-state concurrency, 367–373
should_panic attribute, 226–229
sibling modules, 125
single quote (’), 502
  for characters, 40
  for lifetime parameter names, 205
  for loop labels, 55
?Sized, 445
Sized trait, 445–446, 448
slice type, 77–83
  of array, 83
  string slices, 79–82, 152–153
smart pointer, 315–351
snake case, 44
Software Package Data Exchange (SPDX), 305
speed, xxvii
square brackets ([]), 500
  for array creation, 41
  in the array type, 42
  for element access, 42, 143–145
stack
  and the heap, 60–61
  last in, first out ordering, 60
  popping off of, 60
  pushing onto, 60
standard error (stderr), 270–272
standard output (stdout), 270–272
statements, 46–47
state objects, 384–385
state pattern, 384–393
statically typed, 36
static dispatch, 384
‘static lifetime, 212–213, 428, 476
static variables, 427–428
status line, 465
stderr (standard error), 270–272
stdin function, 16–17
stdout (standard output), 270–272
&str (string slice type), 79–82
stream, 461–464
stringify! macro, 456
string literal, 62
  storage in the binary of, 63
  of string slice type, 81
string slice type (&str), 79–82
String type, 62–64, 147–154
  as_bytes method on, 77
  bytes method on, 153
  chars method on, 153
  concatenation with +, 149–150
  from function on, 63, 148
  indexing into, 151–152
  internal structure of, 63–65, 151–152
  iterating over, 153–154
  len method on, 78
  new function on, 147–148
  parse method on, 26
  push method on, 149
  push_str method on, 63, 149
  slicing, 152–153
  trim method on, 25–26
  UTF-8 encoding of, 147–148, 152–154
Stroustrup, Bjarne, 293
structs, 85–102
  defining, 86
  destructuring, 407–411
  field init shorthand, 87–88
  fields, 86
structs (continued)
   instantiating, 86
   making public, 130–131
   ownership of data, 90–91
   tuple, 89
   unit-like, 89–90
   update syntax, 88–89
students, xxvi
subtraction, 39
super keyword, 125, 130
supertraits, 437–439
symbols, 502–505
syn crate, 454–455
Sync trait, 373–374, 429
T
TCP (Transmission Control Protocol), 460
teams of developers, xxvi
test attribute, 217
test double, 336
test-driven development (TDD), 259–265
test functions, 216–219
tests, 215–241
   custom failure messages for, 224–226
documentation, 298–299
   filtering, 233–235
   ignoring, 235–236
   integration, 236–241
   organizing, 236–241
   of private functions, 237
   running, 230–236
   unit, 236–237
   using Result<T, E> in, 230
   writing, 216–230
thread pool, 472–493
threads, 354–374
   creating with spawn, 355–356,
      473–486
   joining, 356–358
   pausing with sleep, 356
   running closures in, 355–356,
      358–361
thunk, 441–442
Tom’s Obvious, Minimal Language (TOML), 8
to_string method, 148, 200–201
ToString trait, 200–201
trait bounds, 197, 201, 213–214
   conditionally implementing
      methods with, 200–201
trait objects, 379–384, 448
dynamic dispatch, 384
traits, 192–201
   associated types in, 430–431
   default implementations of, 195–197
defining, 192–193
derived, 94–96
   implementing, 193–195
   as parameters, 197–201
   supertraits, 437–439
   unsafe, 429
Transmission Control Protocol, 460
transmitter, 361–366
trim method, 25–26
tuple data type, 40–41
tuples, destructuring, 411
tuple structs, 89, 439–440
two’s complement wrapping, 38
type alias, 440–443, 484
type annotation, 26, 36
type inference, 25
type suffixes, 37
underscore (_), 502
   as a catch-all pattern, 29, 115–116,
      411–413
   in filenames, 5
   as a visual separator in integer
      literals, 37
Unicode scalar value, 40, 152–154
uniform resource identifier, 465
uniform resource locator, 465
unions, 429
unit-like structs, 89–90
unit tests, 236–237
unit type, 41
unrecoverable errors, 161–165
unrolling, 294
unsafe, 420–429
   functions, 423–427
   superpowers, 420–421, 429
   traits, 429
unsized type, 444–446
unwinding, 162
unwrap method, 168
unwrap_or_else method, 255
URI (uniform resource identifier), 465
URL (uniform resource locator), 465
use keyword, 132–138
    and as, 135
    and external packages, 136–137
    and the glob operator, 138
    and nested paths, 137
    and pub, 135–136
user input, 16–17
usize type
    architecture dependent size of, 37
    indexing collection with, 38
    UTF-8 encoding, 147–148, 152–154
V
variables, 32–36
    vs. constants, 33–34
    creating with patterns, 401–402
    global, 427–428
    mutability, 32–33
    shadowing, 34–36
    static, 427–428
    storing values in, 16
variants, 104
vec! macro, 142
vector. See Vec<T> type
Vec<T> type, 142–147
    get method on, 143–145
    indexing into, 143–145
    iterating over, 145
    new function on, 142
    push method on, 142–143
    storing multiple types in, 145–146
vertical pipe (|)
    in closure definitions, 276, 502
    in patterns, 406, 501
Visual Studio, 3
Visual Studio Code, 514
Vlissides, John, 376
W
warnings, 512–513
Weak<T> type, 348–351
web server project, 459–493
where clause, 198
while let loop, 400
while loop, 56–57
Windows installation of Rust, 3
Wirth, Lukas, 451
workspaces, 307–312
wrapping_ * methods, 38
Y
yanking, 306
Z
zero-cost abstractions, 293–294
zero-overhead, 293