

INDEX

Symbols

- & operator, 47
- \$\$ special variable, 76
- * (splat) operator, 47

A

- abstract syntax tree (AST), 23–29, 32–44.
See also AST nodes
- algorithm
 - constant lookup, 162, 163–164
 - Immix, 315
 - LALR parse, 13–19
 - method lookup, 138–151
 - semi-space, 311–312
- allocator pointer, 126
- ancestors method, 259
- Appleby, Austin, 183
- args_add_block AST node, 25
- arguments
 - to a block, 96
 - default values for, 47
 - keyword
 - compiling, 49
 - exploring how Ruby implements, 99–103
 - method, 70–71
 - optional, 48, 96
 - preparing, for normal Ruby methods, 95
 - unnamed, 47, 96
- ARGV array, 65, 75, 79, 262
- Array (C++ class), 287
- Array class, 294–291
- arrays, in Rubinius and MRI, 284–287
- Array#sample method, 260–263
- Array#shift method, 288–291
- AST (abstract syntax tree), 23–29, 32–44.
See also AST nodes
- AST nodes, 23–29, 32–44
 - args_add_block, 25
 - binary, 27–28
 - NEW_CALL, 22–23

- NODE_CALL, 23, 26, 35–43
- NODE_DVAR, 41
- NODE_FCALL, 34–38, 41–43
- NODE_ITER, 39–41
- NODE_SCOPE, 34–44, 46
- attr_accessor method, 98–99, 116
- attribute get and set methods, 94, 98–99
- attribute names, 116–119
- attribute names table, 125, 127
- attr_reader method
 - calling, 97–98
 - optimization by method dispatch, 98–99
- ATTRSET methods, 94, 98–99
- attr_writer method
 - calling, 97–98
 - optimization by method dispatch, 98–99
- autoload keyword, 164

B

- backtraces, in Rubinius and MRI, 281–284
- Baker, Henry, 309
- BasicObject class, 259
- benchmarking Ruby versions, 65–67
- binary AST node, 27–28
- bin density (in a hash table), 175
- Binding class, 208
- binding keyword, 238
- bins (in a hash table), 169
- Bison, 12, 22–23
- bitmap marking, 299–300
- Blackburn, Stephen M., 324
- block arguments, 96
- blocks
 - calling a method with, 71–72
 - as closures, 192–198
 - calling, 61–62, 194–196
 - compiling calls to, 38–44
 - lexical scope for, representation by Ruby, 244–245
 - vs. while loops, speed of, 200–203

- BMETHOD methods, 94
- brace_block grammar rule, 21
- branchless YARV instruction, 85
- built-in Ruby methods, calling, 97–99
- bump allocation, 310
- bytecode, 33
- bytecode (JRuby option), 256
- bytecode instructions
 - Java, 254
 - Rubinius, 277, 278–279
- ByteList (Java object), 264

C

- C++, working together with Ruby, 279–280
- C4 (continuously concurrent compacting) collector, 320
- caches, clearing Ruby's method, 143–144
- calling
 - attr_reader, 97–98
 - attr_writer, 97–98
 - blocks, 61–62, 194–196
 - built-in Ruby methods, 97–99
 - eval with binding, 238–240
 - lambda more than once in the same scope, 216–217
 - lambdas, 209–211
 - methods with blocks, 71–72
 - normal Ruby methods, 95–97
- call stack, 56
- catch tables, 88–90
- CFP (current frame pointer), 57, 88, 95
- CFUNC methods, 61, 94, 97
- child grammar rule, 15
- Class (C++ class), 280
- Class (Ruby class), 117
- class << metaprogramming syntax, 225
- class_alloc C function, 155
- classes
 - Array, 284–291
 - BasicObject, 259
 - Binding, 208
 - Enumerator, 284
 - Fixnum, 110
 - Hash, 169
 - included, 137, 152–154
 - Integer, 62, 72, 97, 142–143, 282–284
 - Object, 133, 157–158, 231–232
 - origin, 150
 - Proc, 211–213
 - Ripper, 9–12, 22–29
 - Ruby, implementing with Java classes, 257–259
 - RubyVM::InstructionSequence, 44–45, 51–53
 - seeing methods and submodules, 152–153
 - singleton, 130, 226–227
 - String, 263–271
- class_eval method, 244
- class instance variables, 120–122
- class keyword, 221
- class method, 111
- class methods, 127–130
 - defining
 - using a new lexical scope, 224
 - using an object prefix, 223
 - scope of, 236
- Class.new method, 113
- class pointers, 106, 107, 109
- class scope, 232–234
- class variables, 120–124
- clearing, Ruby's method caches, 143–144
- climbing, environment pointer ladder
 - in C, 74–75
- closures, 191, 197
 - blocks as, 192–198
 - and current value of self, 241–242
 - defining methods with, 246–248
 - and metaprogramming, 236–244
- code
 - how JRuby executes, 255–257
 - how JRuby parses and compiles, 254–255
 - that writes code, 236–238
- codeloader.rb file, 283
- compilation, 31
- compile.c file, 42
- compiling
 - calls to blocks, 38–44
 - keyword arguments, 49
 - optional arguments, 48
 - simple scripts, 34–38
- compilers
 - introduced in Ruby 1.9 and 2.0, 33–34
 - Rubinius, 277, 290
- comptmt grammar rule, 21
- concurrent collector (in the JVM), 320
- concurrent garbage collection, 205, 317
 - in the JVM, 320–321
 - marking objects, 317–318
 - tricolor marking, 319–320
- constants, 124–125
 - creating, for a new class or module, 159–160
 - finding, 156–158, 160–161
 - lookup algorithm, 162, 163–164
 - table, 125
- const_missing method, 164
- const_tbl pointer, 126
- continuously concurrent compacting (C4) collector, 320
- copying, a stack frame to the heap, 208, 209

- copying garbage collection, 205, 309
 - bump allocation, 310–311
 - Eden heap, 312–313
 - and semi-space algorithm, 311–312
- copy-on-write
 - for strings, 265–271
 - in Unix, 300
- core#`define_method` method, 53
- `count_objects` method, 129
- creating
 - constants, for a new class or module, 159–160
 - lambdas, 207–209
 - refinements, 228
 - unique and nonshared strings, 267
- `cref` (C structure), 244, 245
- `cref` pointer, 68, 78
- current frame pointer (CFP), 57, 88, 95

D

- default values, for arguments, 47
- `define_method` method, 246
- defining
 - class methods
 - using a new lexical scope, 224
 - using an object prefix, 223
 - methods
 - alternative ways of, 223–231
 - normal process of, 221–223
 - using singleton classes, 226–227
 - using singleton classes in a lexical scope, 227–228
- `def` keyword, 221–223
- defs/keywords* file, 8
- density (in a hash table), 175
- disadvantages of mark and sweep, 302
- displaying the local table, 51–53
- `--dump parsetree` (Ruby option), 29
- `dup` method, 265–266
- dynamic variable access, 71–74

E

- `each` method, 91
- Eden heap, 312–313
- ensure keyword, 90
- Enumerator class, 284
- environment, 197, 198
- environment pointer (EP), 68, 194, 197
- environment pointer ladder, 74–75
- EP (environment pointer), 68, 194, 197
- `eq?` method, 175
- `eval` method, 78, 236–240
- examples
 - grammar rule, 14
 - `instance_eval`, 240–241
 - method lookup, 139–140
 - `Module#prepend`, 146

- executing
 - calls to blocks, 61–62, 194–196
 - if statements, 84–86
 - simple scripts, 58–60
- expanding hash tables, 174–175
- experiments
 - `Array#shift`, Rubinius implementation of, 288–291
 - backtraces, comparing in MRI and Rubinius, 281–284
 - class methods, where Ruby saves, 127–130
 - closures, defining methods with, 246–248
 - constant, which Ruby finds first, 162–164
 - copy-on-write performance, measuring, 267–271
 - for loops, how Ruby implements, 90–92
 - garbage collection (MRI), seeing in action, 302–308
 - hashes
 - inserting new element into, 177–179
 - retrieving values from, 172–173
 - using with object keys, 183–189
 - instance variable, time required to save, 113–115
 - JIT compiler, monitoring, 260–263
 - keyword arguments, how Ruby implements, 99–103
 - local table, displaying, 51–53
 - local variables, changing after calling `lambda`, 214–217
 - modules, modifying after including, 151–154
 - Ruby scripts
 - parsing, 9–12
 - tokenizing, 9–12
 - Ruby versions, benchmarking, 65–67
 - `self`, how it changes with lexical scope, 231–236
 - special variables, exploring, 75–78
 - verbose GC mode in JRuby, using, 321–324
 - while loops vs. passing blocks, speed of, 200–203
 - YARV instructions, displaying, 44–45
- `extend` method, 138

F

- false value, 110
- Fenichel, Robert R., 309
- finding constants, 156–158, 160–161
- first-class citizen, treating a function as, 203–213

- Fixnum class, 110
- FIXNUM_FLAG value, 110
- for loops, 90–92
- free bitmap, 300
- free lists, 297–298
- from-space, 311
- function, as a first-class citizen, 203–213
- function call, 36
- further reading (on garbage collection), 324

G

- G1 (garbage-first) collector, 320
- garbage collection (GC), 205, 207, 295–296
 - concurrent, 205, 317
 - in the JVM, 320–321
 - marking objects, 317–318
 - tricolor marking, 319–320
 - copying, 205, 309
 - bump allocation, 310–311
 - Eden heap, 312–313
 - and semi-space algorithm, 311–312
 - further reading, 324
 - garbage objects, 297
 - generational. *See* generational garbage collection
 - in JRuby, 309
 - verbose mode, 321–324
 - in the JVM, 320–321
 - live objects, 297
 - mark-and-sweep. *See* mark-and-sweep garbage collection
 - in MRI, 297–302
 - full collection, 304–305
 - interpreting GC profile report, 305–308
 - lazy sweeping, 303–304
 - RVALUE (C structure), 301
 - seeing in action, 302–303
 - purpose of, 207
 - in Rubinius, 309
 - in Ruby 2.1, 309, 315
- garbage-first (G1) collector, 320
- garbage objects, 297, 299
- Garden of Eden, 312–313
- GC. *See* garbage collection
- GC.disable, 113
- GC::Profiler, 306
- GC.start, 304
- generational garbage collection, 205, 313
 - and mature objects, 315
 - promoting objects, 314–315
 - and references, 316
 - and semi-space algorithm, 314
 - weak generational hypothesis, 313–314

- generic_iv_tbl (C structure), 113
- generic objects, 109–110, 111, 113
- GET_EP (C macro), 74
- getlocal YARV instruction,
 - C implementation of, 74–75
- GET_PREV_EP (C macro), 75
- getting class variables, 122
- global method cache, 142
- global variable names, 7
- goto statement, 43
- gperf, 8
- grammar rules, 14, 20, 22
 - Bison, 22
 - brace_block, 21
 - child , 15
 - compstmt, 21
 - example, 14
 - keyword_do, 21
 - method_call, 22
 - opt_block_param, 21
 - primary value, 21
 - top_compstmt, 20
- grammar rule file, 8
- grammar rule stack, 16

H

- Hash class, 169
- Hash#key?, 100
- hash method, 182
- hash tables
 - bin density, 175
 - bins, 169
 - collisions, 174
 - expanding, 174–175
 - functions, 170, 181–183
 - inserting new element into, 177–179
 - key? method, 100
 - optimization in Ruby 2.0, 187–189
 - packed, 188
 - prime numbers for, 180–181
 - rehashing entries, 175–176
 - retrieving values from, 171, 172–173
 - saving values in, 169–172
 - using with object keys, 183–189
- heap, 204, 297
- Hosking, Antony, 324
- hot spots, 261

I

- identifiers, 7
- if...else statements, 84–86
- Immix algorithm, 315
- implementing
 - hash functions, 181–183
 - Module#prepend, 150–151
 - modules, 135–138

- Ruby classes with Java classes, 257–259
- Ruby objects with C++ objects, 280–281
- included classes, 137, 153–154
- include method, 136
- include_modules_at (C function), 155
- including
 - a module into a class, 136–138
 - one module into another, 145–146
 - two modules into one class, 144–145
- inheritance, 118
 - multiple, 141–142
- inline method cache, 143
- insns.def* file, 63
- inspecting klass and ivptr, 107–108
- instance_eval method, 78, 240, 244, 245
 - creating a singleton class for a new lexical scope, 243–244
 - example, 240–241
 - and lexical scope, 243–245
 - and self, 242
- instance-level attribute names, 118
- instance variable arrays, 107, 108
- instance variable counts, 107
- instance variable methods, 94
- instance variable names, 7
- instance variables
 - for generic objects, 111, 113
 - time required to save, 113–115
- instance_variable_set method, 111
- instance_variables method, 111
- instructions, YARV. *See* YARV instructions
- instruction sequence (ISEQ) methods, 93–94, 95
- Integer class, 62, 72, 97, 142–143, 282–284
- integer.rb* file, 283
- Integer#times method, 97, 283
- internal heap structure, 298
- interpreting a GC profile report, 305–308
- ISEQ (instruction sequence) methods, 93–94, 95
- iseq_compile_each (C function), 42
- iterating through the AST, 42–43
- IVAR methods, 94, 98–99
- iv_index_tbl pointer, 125, 126
- ivptr pointer, 107, 109, 114
- iv_tbl pointer, 126

J

- Java bytecode instructions, 254
- java command, to launch JRuby, 253
- Java Virtual Machine (JVM), 253, 256
 - garbage collection, 309, 320–321
- Jay parser, 254

- JIT (just-in-time) compiler, 255, 260, 262–263
- Jones, Richard, 324
- JRuby, 251
 - code execution, 255–257
 - code parsing and compiling, 254–255
 - garbage collection, 309, 321–324
 - and MRI, saving string data with, 264–265
 - vs. MRI performance, 263
- jruby.jar* file, 253
- jumping from one scope to another, 86–90
- jump YARV instruction, 85
- just-in-time (JIT) compiler, 255, 260, 262–263
 - J-verbose:gc (JRuby option), 321
- JVM (Java Virtual Machine), 253, 256
 - garbage collection, 309, 320–321
 - J-XX:+PrintCompilation (JRuby option), 261

K

- key? method, 100
- keyword arguments
 - compiling, 49
 - exploring how Ruby implements, 99–103
- keyword_do
 - grammar rule, 21
 - token, 7
- keyword_end token, 21
- keywords
 - autoload, 164
 - binding, 238
 - class, 221
 - def, 221–223
 - ensure, 90
 - lambda
 - calling more than once in the same scope, 216–217
 - changing local variables after calling, 214–217
 - next, 90
 - redo, 90
 - rescue, 90
 - retry, 90
 - return, 90
 - TOPLEVEL_BINDING, 194
 - unless, 85
- klass pointer, 106, 107, 109, 199

L

- ladder, environment pointer, 74–75
- LALR (Look-Ahead Left Reversed Rightmost Derivation)
 - parse algorithm, 13–19
 - state table, 18

- lambda keyword
 - calling more than once in the same scope, 216–217
 - changing local variables after calling, 214–217
 - lambdas, 197, 198, 203–213
 - calling, 209–211
 - creating, 207–209
 - Landin, Peter J., 191
 - lazy sweeping, 300–301, 303–304
 - leave YARV instruction, 45, 60
 - Lex, 8
 - lexical scope, 68, 78, 158
 - for blocks, 244–245
 - and instance_eval, 243–245
 - and self, 231–236
 - Lins, Rafael D., 324
 - Lisp, 191–192, 197–198, 295, 297, 324
 - live objects, 297, 299
 - LLVM (Low-Level Virtual Machine), 277
 - loader.rb file, 283
 - locality of reference, 310
 - local table, 46–53
 - local variable access, 67–70
 - local variables, changing after calling
 - lambda, 214–217
 - look ahead, 18
 - Look-Ahead Left Reversed Rightmost Derivation (LALR)
 - parse algorithm, 13–19
 - state table, 18
 - lookup algorithm, for constants, 162, 163–164
 - Low-Level Virtual Machine (LLVM), 277
- ## M
- m_tbl pointer, 125, 126
 - major collections, 315, 323–324
 - mark-and-sweep garbage collection, 205, 297–302
 - disadvantages of, 302
 - free list, 297–298
 - and JRuby, 309
 - lazy sweeping, 300–301
 - marking objects, 299–300
 - and Rubinius, 309
 - seeing in action, 302–308
 - sweeping, 300
 - marking objects, 299–300
 - tricolor, 319–320
 - while the object graph changes, 317–319
 - mark stack, 319
 - Matsumoto, Yukihiro “Matz”, 4, 251
 - Matz’s Ruby Interpreter. *See* MRI
 - McCarthy, John, 191, 295, 324
 - McKinley, Kathryn S., 324
 - metaclass, 129, 226
 - metaclass scope, 235
 - metaprogramming, and closures, 236–244
 - method arguments, 70–71
 - method caches
 - clearing Ruby’s, 143–144
 - global, 142
 - inline, 143
 - method_call grammar rule, 22
 - method dispatch, 84, 92–93
 - optimizing attr_reader and attr_writer, 98–99
 - method lookup, 92–93, 138
 - algorithm, 138–151
 - example, 139–140
 - methods
 - acting as closures, 247–248
 - ancestors, 259
 - Array#sample, 260–263
 - Array#shift, 288–291
 - attr_accessor, 98–99, 116
 - attribute get and set, 94, 98–99
 - attr_reader
 - calling, 97–98
 - optimization by method dispatch, 98–99
 - attr_writer
 - calling, 97–98
 - optimization by method dispatch, 98–99
 - calling
 - with blocks, 71–72
 - built-in, 97–99
 - normal, 95–97
 - class, 111
 - class_eval, 244
 - class methods, 127–130
 - scope of, 236
 - Class.new, 113
 - const_missing, 164
 - core#define_method, 53
 - count_objects, 129
 - def, 221–223
 - define_method, 246
 - definition process
 - alternative, 221–231
 - normal, 221–223
 - dup, 265–266
 - each, 91
 - eq!?, 175
 - eval, 78, 236–240
 - extend, 138
 - hash, 182
 - include, 136
 - instance_eval. *See* instance_eval method

- instance_variables, 111
- instance_variable_set, 111
- Integer#times, 97, 283
- key?, 100
- module_eval, 244
- Module.nesting, 231
- Module#prepend, 146
- ObjectSpace#count_objects, 129, 302
- Ripper.lex, 9
- Ripper.sexp, 23
- Rubinius.primitive, 280
- set_trace_func, 45, 58
- source_location, 288
- String#[], 280
- String#upcase, 97
- Struct#each, 97
- using, 229
- method tables, 116, 117, 125, 126, 153
- method types, 93–95
 - ATTRSET, 94, 98–99
 - BMETHOD, 94
 - CFUNC, 61, 94, 97
 - ISEQ, 93–94, 95
 - IVAR, 94, 98–99
 - MISSING, 95
 - normal, 95–97
 - NOTIMPLEMENTED, 94
 - OPTIMIZED, 94
 - REFINED, 95
 - UNDEF, 94
 - ZSUPER, 94
- Miniruby, 63
- minor collections, 315, 322
- Minsky, Marvin, 309
- MISSING methods, 95
- modifying
 - Array#shift, 289–291
 - modules, after including, 151–154
 - shared strings, 270–271
- module_eval method, 244
- Module.nesting method, 231
- Module#prepend method, 146, 150–151
- modules
 - constants, creating for, 159–160
 - how Ruby copies, 154–155
 - implementing as classes, 135–138
 - including
 - into a class, 136–138
 - one into another, 145–146
 - two into one class, 144–145
 - modifying after including, 151–154
- Moore, Peter, 172
- Moss, Eliot, 324
- MRI (Matz’s Ruby Interpreter), 4, 251
 - arrays in Rubinius and, 284–287
 - backtraces, 281–282
 - free lists, use of multiple, 298
 - garbage collection, 297–308

- running programs, with JRuby, 252–259
- strings, in JRuby and, 263–267
- multiple inheritance, 141–142
- MurmurHash, 183

N

- nd_class pointer, 159, 221
- nd_next pointer, 159
- nd_refinements pointer, 230
- new object lifetime, 314
- NEW_CALL AST node, 22–23
- next keyword, 90
- nil value, 110
- NODE_CALL AST node, 23, 26, 35–43
- NODE_DVAR AST node, 41
- NODE_FCALL AST node, 34–38, 41–43
- NODE_ITER AST node, 39–41
- NODE_SCOPE AST node, 34–44, 46
- Noria, Xavier, 151
- normal methods, calling, 95–97
- NOTIMPLEMENTED methods, 94
- nth back reference, special variables, 80
- numiv (C structure value), 107, 109
- Nutter, Charles, 321

O

- Object (C++ class), 281
- Object class, 133, 157–158, 231–232
- object graph, 299
- object prefix, metaprogramming
 - syntax, 223
- objects, 106–113
 - generic, 109–110, 111, 113
 - as hash keys, 183
 - Ruby, implementing with C++
 - objects, 280–281
- ObjectSpace#count_objects method, 129, 302
- :on_ident (Ripper output value), 10
- :on_int (Ripper output value), 10
- operand optimization, 69
- opt_block_param grammar rule, 21
- OPTIMIZED methods, 94
- optional arguments, 48, 96
- opt_lt YARV instruction, 85
- opt_plus YARV instruction, 38, 60
- opt_send_simple YARV instruction, 38, 60
- origin class, 150
- origin pointer, 126

P

- packed hashes, 188
- parallel collector (in the JVM), 320
- parent namespace, finding a constant in, 157–158

- parse.c* file, 13
- parse.y* file, 8, 12, 19, 20, 22, 79
- parser generator, 12
- parser_yylex (C function), 8, 79
- parsetree (Ruby option), 29
- parsing, 3, 4, 9–12
- passing a block to each, 200
- PC (program counter), 57
- permanent generation, 315
- pointers
 - class, 106, 107, 109
 - CFP, 57, 88, 95
 - climbing the environment pointer
 - ladder in C, 74–75
 - const_tbl, 126
 - cref, 68, 78
 - EP, 68, 194, 197
 - iv_index_tbl, 125, 126
 - ivptr, 107, 109, 114
 - iv_tbl, 126
 - klass, 106, 107, 109, 199
 - m_tbl, 125, 126
 - nd_class, 159, 221
 - nd_next, 159
 - nd_refinements, 230
 - origin, 126
 - PC, 57
 - refined_class, 126
 - self, 36, 57, 199, 231
 - SP, 57
 - super, 119, 125, 126
 - svar, 75
 - svar/cref, 68
 - VALUE, 106, 107, 110, 204
- preparing arguments for normal Ruby
 - methods, 95
- primary value grammar rule, 21
- prime numbers, for hash tables, 180–181
- problems, solved by garbage collectors, 297
- Proc class, 211–213
- procs, 203, 208, 211–213
- program counter (PC), 57
- program grammar rule, 20
- programs, running with MRI and JRuby, 252–259
- promoting objects, 314
- putobject YARV instruction, 59
- putself YARV instruction, 36, 58, 63

R

- RArray (C structure), 109, 110, 285, 286
- RBasic (C structure), 106, 107, 109, 110, 112
- rb_binding_t (C structure), 239
- rb_block_t (C structure), 72, 192, 196, 197, 198–199

- rb_classext_struct (C structure), 125, 126
- rb_control_frame_t (C structure), 57, 194, 198–199
- rb_env_t (C structure), 208, 209, 239
- rb_include_class_new (C function), 154
- rb_proc_t (C structure), 208, 209, 212
- rb_reserved_word (C function), 8
- rbx command, 274, 278
- RClass (C structure), 108, 115, 125–126, 127
- RCLASS_CONST_TBL (C macro), 155
- RCLASS_IV_TBL (C macro), 155
- RCLASS_M_TBL (C macro), 155
- RCLASS_ORIGIN (C macro), 155
- reading
 - Array#shift, 288
 - a Bison grammar rule, 22
 - the RBasic and RObject definitions, 112
 - the RClass definition, 127
- receiver-arguments-message pattern, 36, 37, 42, 59
- redo keyword, 90
- reduce (parsing operation), 17
- references between generations, 316
- referencing environment, 197, 198
- refined_class pointer, 126
- REFINED methods, 95
- refinements, 228–231
- regular expressions, for special
 - variables, 80
- rehash (C function), 176
- rehashing hash table entries, 175–176
- rescue keyword, 90
- reserved word, 7
- retrieving a value, from a hash table, 171
- retry keyword, 90
- return keyword, 90
- RHash (C structure), 169
- Ripper, 9–12, 22–29
- Ripper class, 9–12, 22–29
- Ripper.lex method, 9
- Ripper.sexp method, 23
- RObject (C structure), 106, 107, 109, 112
- root object, 299
- RRegExp (C structure), 109, 110
- RString (C structure), 109, 110, 205, 206, 264
- RTypedData (C structure), 212
- Rubinius, 273
 - backtraces, 282–284
 - bytecode instructions, 277, 278–279
 - compiler, 277, 290
 - garbage collection, 309
 - kernel and virtual machine, 274–281
- Rubinius.primitive method, 280
- Ruby, working together with C++, 279–280
- Ruby 2.1, garbage collection, 309, 315

- RubyBasicObject (Java class), 259
- RubyClass (Java class), 258
- ruby --dump parsetree option, 29
- RubyFixnum (Java class), 257
- Ruby hash (RHash), C structure, 169
- RubyIO (Java class), 257
- RubyModule (Java class), 259
- RubyObject (Java class), 258
- ruby_sourceline variable, 23
- RubyString (Java class), 264
- Ruby versions, benchmarking, 65–67
- RubyVM::InstructionSequence class, 44–45, 51–53
- ruby -y option, 19
- running programs, with MRI and JRuby, 252–259
- RVALUE (C structure), 298, 301

S

- Sasada, Koichi, 33, 55
- saving
 - array data, with Rubinius and MRI, 285–291
 - instance variables for generic objects, 113
 - string data, with JRuby and MRI, 264–265
 - string values, 204–207
 - values in a hash table, 169–171
- Scheme, 192, 197
- scope, 35. *See also* lexical scope
 - jumping from one to another, 86–90
- scripts
 - compiling simple, 34–38
 - executing, 58–60
 - parsing, 9–12
- self object
 - in a class method, 234
 - in a class scope, 232
 - and closures, 241–242
 - and lexical scope, 231–236
 - in a metaclass scope, 233
 - in the top scope, 231
- self pointer, 36, 57, 199, 231
- semi-space algorithm, 311–312
- send YARV instruction, 92–95, 247
- serial collector (in the JVM), 320
- setting class variables, 122
- set_trace_func method, 45, 58
- setlocal YARV instruction, 69
- shift (parsing operation), 16
- simple values, 37, 110–111
- singleton classes, 130, 226–227
- source_location method, 288
- SP (stack pointer), 57
- specialized instructions, 38

- special value, 71
- special variables, 68, 75–80
 - \$\$, 76
 - definitive list of Ruby's, 79
 - nth back reference, 80
 - for regular expressions, 80
- speed
 - of JRuby programs with JIT, 262–263
 - of Ruby versions, 65–67
 - of while loops vs. passing blocks, 200–203
- splat (*) operator, 47
- splat argument array, 96
- st_table (C structure), 169
- st_table_entry (C structure), 169
- stack, 204
- stack frame, copying to the heap, 205, 209
- stack-oriented virtual machines, 35
- stack pointer (SP), 57
- state table, 18
- Steele, Guy, 192, 197
- String#[] method, 280
- String class, 263–271
- string.rb file, 280
- strings
 - creating unique and nonshared, 267
 - in JRuby and MRI, 263–267
- String#upcase method, 97
- string values, how Ruby saves, 204–207
- Struct#each method, 97
- submodules, 152
- super pointer, 119, 125, 126
- SUPPORT_JOKE (C source code value), 43
- survivor spaces, 314
- Sussman, Gerald, 192, 197
- svar pointer, 75
- svar/cref pointers, 68
- sweeping, 300–301

T

- tenured generation, 315
- throw YARV instruction, 87
- tIDENTIFIER token, 7
- time (Unix command), 65
- tINTEGER token, 6
- tokenization, 3, 9–12
- tokens, 4–9
- top scope, 233
- top self object, 36, 59, 232
- top_compstmt grammar rule, 20
- TOPLEVEL_BINDING keyword, 194
- to-space, 311
- trace YARV instruction, 45
- tricolor marking, 319–320
- triggering major collections, 323–324
- true value, 110

Tuple (C++ class), 287
types of Ruby methods, 93–95

U

UNDEF methods, 94
unless keyword, 85
unnamed arguments, 47, 96
until...end loop, 85
using method, 229

V

VALUE pointer, 106, 107, 110, 204
values
 array, how Ruby saves, 285–291
 default, for arguments, 47
 expanding hash tables to
 accommodate more, 174–175
 false, 110
 FIXNUM_FLAG, 110
 nil, 110
 simple, 110–111
 special, 71
 string, how Ruby saves, 204–207,
 263–271
 true, 110
variable access
 dynamic, 71–74
 local, 67–70
variables, 67–74
 class, 120–124
 class instance, 120–122
 instance
 for generic objects, 111, 113
 time required to save, 113–115
 local, changing after calling `lambda`,
 214–217
 special. *See* special variables
visualizing copy-on-write, 269–270
visualizing two instances of one class, 108
vm_core.h file, 198
vm_exec.c file, 64
vm_getivar (C function), 99
vm.inc file, 63
vm_inshelper.h file, 74
VM_METHOD_TYPE_ISEQ (C source code
 value), 95
VM_METHOD_TYPE_CFUNC (C source code
 value), 97
VM_METHOD_TYPE_REFINED (C source code
 value), 229
vm_setivar (C function), 98

W

weak generational hypothesis, 313–314
while...end loop, 85, 200–203
write barriers, 316

X

-Xbootclasspath (Java option), 253

Y

Yacc (Yet Another Compiler
 Compiler), 12
YARV (Yet Another Ruby Virtual
 Machine), 33, 56–62
YARV instructions, 34, 63–64
 branchunless, 85
 displaying, 44–45
 getlocal, 74–75
 jump, 85
 leave, 45, 60
 opt_lt, 85
 opt_plus, 38, 60
 opt_send_simple, 38, 60
 putobject, 59
 putsself, 36, 58, 63
 send, 92–95, 247
 setlocal, 69
 taking a close look at, 63–64
 throw, 87
 trace, 45
Yet Another Compiler Compiler
 (Yacc), 12
Yet Another Ruby Virtual Machine
 (YARV), 33, 56–62
Yochelson, Jerome C., 309
-y (Ruby option), 19

Z

ZSUPER methods, 94