# Index

> Any inaccuracies in this index may be explained by the fact that it has been prepared with the help of a computer.
>
> Donald E. Knuth, *Fundamental Algorithms* (Volume 1 of *The Art of Computer Programming*)

Page numbers for code definitions are in italics.
Page numbers followed by *n* indicate footnotes.


`*` (primitive multiplication procedure), *5*
`+` (primitive addition procedure), *5*
`-` (primitive subtraction procedure), *6*
  as negation, *18 n*
`/` (primitive division procedure), *6 18*
`=` (primitive numeric equality predicate), *18*
`=number?`, *150*
`=zero?` (generic), 193 (ex. 2.80)
  for polynomials, 209 (ex. 2.87)
`<` (primitive numeric comparison predicate),
`>` (primitive numeric comparison predicate), *18*
`>=`, *20*
`;`. *See* semicolon
`!` in names, 220 *n*
`?`, in predicate names, 24 *n*
`"` (double quote), 143 *n*
`'` (single quote), 143
  `read` and, 383 *n*, 485 *n*
`` ` `` (backquote), 575 *n*
`,` (comma, used with backquote), 575 *n*
`#f`, *18 n*
`#t`, *18 n*
$\mapsto$ notation for mathematical function, 69 *n*
$\lambda$ calculus. *See* lambda calculus
$\pi$. *See* pi
$\sum$ (sigma) notation, 58
$\theta$. *See* theta
Abelson, Harold, 3 *n*
`abs`, *17*, *18*
absolute value, 17
abstract data, 83. *See also* data abstraction
abstraction. *See also* means of abstraction; data abstraction; higher-order procedures
  common pattern and, 58
  metalinguistic, 360
  procedural, 26
  in register-machine design, 499–502
  of search in nondeterministic programming, 418
abstraction barriers, 81, 87–90, 169
  in complex-number system, 170
  in generic arithmetic system, 188
abstract models for data, 91 *n*
abstract syntax
  in metacircular evaluator, 364
  in query interpreter, 469
`accelerated-sequence`, *337*
`accumulate`, 61 (ex. 1.32), *116*
  same as `fold-right`, 121 (ex. 2.38)
`accumulate-n`, 120 (ex. 2.36)
accumulator, 114, 224 (ex. 3.1)
Áchárya, Bháscara, 42 *n*
Ackermann's function, 36 (ex. 1.10)
acquire a mutex, 311
actions, in register machine, 498–499
`actual-value`, *402*

Ada, 453 (ex. 4.63)
  recursive procedures, 35
Adams, Norman I., IV, 394 *n*
`add` (generic), *189*
  used for polynomial coefficients, 206
`add-action!`, 276, *280*
`add-binding-to-frame!`, *378*
`add-complex`, *173*
`add-complex-to-schemenum`, *194*
`add-interval`, *94*
`add-lists`, *410*
`add-poly`, *204*
`add-rat`, *84*
`add-rule-or-assertion!`, *481*
`add-streams`, *329*
`add-terms`, *206*
`add-to-agenda!`, 280, *284*
`add-vect`, 136 (ex. 2.46)
`addend`, *148*
adder
  full, 275
  half, 274
  ripple-carry, 278 (ex. 3.30)
`adder` (primitive constraint), *290*
additivity, 82, 170, 179–186, 191
address, 534
address arithmetic, 534
Adelman, Leonard, 53 *n*
`adjoin-arg`, *551 n*
`adjoin-set`, 152
  binary-tree representation, *157*
  ordered-list representation, 155 (ex. 2.61)
  unordered-list representation, *152*
  for weighted sets, *167*
`adjoin-term`, 205, *209*
`advance-pc`, *525*
`after-delay`, 277, *281*
agenda. *See* digital-circuit simulation
A'h-mose, 47 *n*
algebra, symbolic. *See* symbolic algebra
algebraic expression, 202
  differentiating, 145–151
  representing, 147–151
  simplifying, 149–150
algebraic specification for data, 91 *n*
Algol
  block structure, 31
  call-by-name argument passing, 324 *n*, 402 *n*
  thunks, 324 *n*, 401 *n*
  weakness in handling compound objects, 296 *n*
algorithm
  optimal, 119 *n*
  probabilistic, 52–53, 216 *n*
aliasing, 234 *n*
`all-regs` (compiler), *587 n*
Allen, John, 541 *n*
alternative of `if`, 19
`always-true`, *473*
`amb`, *414*
`amb` evaluator. *See* nondeterministic evaluator
`ambeval`, *429*
`an-element-of`, *414*
`an-integer-starting-from`, *414*
analog computer, 347 (fig. 3.34)
`analyze`
  metacircular, *395*
  nondeterministic, 428
`analyze-...`
  metacircular, 395–397, 398 (ex. 4.23)
  nondeterministic, 429–433
`analyze-amb`, *434*
analyzing evaluator, 393–398
  as basis for nondeterministic evaluator, 426
  `let`, 398 (ex. 4.22)
`and` (query language), 446
  evaluation of, 456, 471, 488 (ex. 4.76)
`and` (special form), *19*
  evaluation of, 19
  why a special form, 19
  with no subexpressions, *374* (ex. 4.4)
and-gate, 273
  `and-gate`, *277*
`angle`
  data-directed, *184*
  polar representation, *175*
  rectangular representation, *174*
  with tagged data, *178*

`angle-polar`, *177*
`angle-rectangular`, *176*
`announce-output`, *383*
APL, 118 *n*
Appel, Andrew W., 586 *n*
`append`, 102, *103*, *255* (ex. 3.12)
  as accumulation, 119 (ex. 2.33)
  `append!` vs., 255 (ex. 3.12)
  with arbitrary number of arguments, 590 *n*
  as register machine, 539 (ex. 5.22)
  "what is" (rules) vs. "how to" (procedure), 439–440
`append!`, *255* (ex. 3.12)
  as register machine, 539 (ex. 5.22)
`append-instruction-sequences`, 572, *589*
`append-to-form` (rules), *451*
`application?`, *372*
applicative-order evaluation, 16
  in Lisp, 17
  normal order vs., 21 (ex. 1.5), 49 (ex. 1.20), 399–401
`apply` (lazy), *403*
`apply` (metacircular), *366*
  primitive `apply` vs., 387 *n*
`apply` (primitive procedure), *183 n*
`apply-dispatch`, *553*
  modified for compiled code, *604*
`apply-generic`, *184*
  with coercion, *196*, 200 (ex. 2.81)
  with coercion by raising, 201 (ex. 2.84)
  with coercion of multiple arguments, 201 (ex. 2.82)
  with coercion to simplify, 201 (ex. 2.85)
  with message passing, *187*
  with tower of types, 198
`apply-primitive-procedure`, 366, 377, *382*
`apply-rules`, *476*
arbiter, 313 *n*
arctangent, 174 *n*
`argl` register, 548
argument(s), 6
  arbitrary number of, 6, 104 (ex. 2.20)
  delayed, 347
argument passing. *See* call-by-name argument passing; call-by-need argument passing
Aristotle's *De caelo* (Buridan's commentary on), 313 *n*
arithmetic
  address arithmetic, 534
  generic, 187 (*see also* generic arithmetic operations)
  on complex numbers, 170
  on intervals, 93–97
  on polynomials (*see* polynomial arithmetic)
  on power series, 333 (ex. 3.60), 334 (ex. 3.62)
  on rational numbers, 83–87
  primitive procedures for, 5
`articles`, *421*
ASCII code, 161
`assemble`, *520*, *521 n*
assembler, 515, 520–523
`assert!` (query interpreter), 462
assertion, 441
  implicit, 449
`assign` (in register machine), 497
  simulating, 524
  storing label in register, 506
`assign-reg-name`, *524*
`assign-value-exp`, *524*
assignment, 218–236. *See also* `set!`
  benefits of, 225–229
  bugs associated with, 234 *n*, 235
  costs of, 229–236
`assignment?`, *369*
`assignment-value`, *369*
`assignment-variable`, *369*
assignment operator, 219. *See also* `set!`
`assoc`, *268*
`atan` (primitive procedure), *174 n*
atomic operations supported in hardware, 313 *n*
atomic requirement for `test-and-set!`, 312
`attach-tag`, *176*
  using Scheme data types, 193 (ex. 2.78)
`augend`, *148*

automagically, 416
automatic search, 412. *See also* search
  history of, 416 *n*
automatic storage allocation, 534
`average`, *23*
`average-damp`, *72*
average damping, 70
`averager` (constraint), 295 (ex. 3.33)

backquote, 575 *n*
backtracking, 416. *See also* nondeterministic computing
Backus, John, 356 *n*
Baker, Henry G., Jr., 541 *n*
balanced binary tree, 157. *See also* binary tree
balanced mobile, 111 (ex. 2.29)
bank account, 219, 251 (ex. 3.11)
  exchanging balances, 308
  joint, 233, 236 (ex. 3.7)
  joint, modeled with streams, 356 (fig. 3.38)
  joint, with concurrent access, 298
  password-protected, 225 (ex. 3.3)
  serialized, 305
  stream model, 354
  transferring money, 310 (ex. 3.44)
barrier synchronization, 315 *n*
Barth, John, 359
Basic
  restrictions on compound data, 99 *n*
  weakness in handling compound objects, 296 *n*
Batali, John Dean, 547 *n*
`begin` (special form), *220*
  implicit in consequent of `cond` and in procedure body, 221 *n*
`begin?`, *371*
`begin-actions`, *371*
`below`, 128, 140 (ex. 2.51)
Bertrand's Hypothesis, 330 *n*
`beside`, 128, *139*
bignum, 536
binary numbers, addition of. *See* adder
binary search, 155
binary tree, 155
  balanced, 157
  converting a list to a, 159 (ex. 2.64)
  converting to a list, 158 (ex. 2.63)
  for Huffman encoding, 162
  represented with lists, 156
  sets represented as, 155–160
  table structured as, 272 (ex. 3.26)
bind, 28
binding, 237
  deep, 380 *n*
binomial coefficients, 42 *n*
black box, 26
blocked process, 312 *n*
block structure, 30–31, 388–390
  in environment model, 249–251
  in query language, 490 (ex. 4.79)
body of a procedure, 12
Bolt Beranek and Newman Inc., 3 *n*
Borning, Alan, 286 *n*
Borodin, Alan, 119 *n*
bound variable, 28
box-and-pointer notation, 97
  end-of-list marker, 99
branch of a tree, 9
`branch` (in register machine), 496
  simulating, 525
`branch-dest`, *526*
breakpoint, 532 (ex. 5.19)
broken heart, 542
B-tree, 158 *n*
bug, 1
  capturing a free variable, 29
  order of assignments, 235
  side effect with aliasing, 234 *n*
bureaucracy, 463
Buridan, Jean, 313 *n*
busy-waiting, 312 *n*

C
  compiling Scheme into, 610 (ex. 5.52)
  error handling, 565 *n*, 607 *n*
  recursive procedures, 35
  restrictions on compound data, 99 *n*
  Scheme interpreter written in, 610 (ex. 5.51), 610 (ex. 5.52)
`ca...r`, 100 *n*

cache-coherence protocols, 299 *n*
`cadr`, 100 *n*
calculator, fixed points with, 69 *n*
call-by-name argument passing, 324 *n*, 402 *n*
call-by-need argument passing, 324 *n*, 402 *n*
  memoization and, 332 *n*
`call-each`, *279*
cancer of the semicolon, 11 *n*
canonical form, for polynomials, 211
capturing a free variable, 29
`car` (primitive procedure), *85*
  axiom for, 91
  implemented with vectors, 537
  as list operation, 100
  origin of the name, 85 *n*
  procedural implementation of, *91*, 92 (ex. 2.4), *260*, *409*
Carmichael numbers, 53 *n*, 55 (ex. 1.27)
case analysis
  data-directed programming vs., 366
  general, 17 (*see also* `cond`)
  with two cases (`if`), 19
`cd...r`, 100 *n*
`cdr` (primitive procedure), *85*
  axiom for, 91
  implemented with vectors, 537
  as list operation, 100
  origin of the name, 85 *n*
  procedural implementation of, *91*, 92 (ex. 2.4), *260*, *409*
`cdr` down a list, 101
cell, in serializer implementation, 311
`celsius-fahrenheit-converter`, *287*
  expression-oriented, *296* (ex. 3.37)
`center`, *95*
Cesàro, Ernesto, 226 *n*
`cesaro-stream`, *353*
`cesaro-test`, *227*
Chaitin, Gregory, 226 *n*
Chandah-sutra, 46 *n*
change and sameness
  meaning of, 232–234
  shared data and, 257
changing money. *See* counting change
chaos in the Solar System, 3 *n*
Chapman, David, 416 *n*
character, ASCII encoding, 161
character strings
  primitive procedures for, 485 *n*, 578 *n*
  quotation of, 143 *n*
Charniak, Eugene, 416 *n*
Chebyshev, Pafnutii L'vovich, 330 *n*
chess, eight-queens puzzle, 124 (ex. 2.42), 420 (ex. 4.44)
chip implementation of Scheme, 547, 548 (fig. 5.16)
chronological backtracking, 416
Church, Alonzo, 63 *n*, 93 (ex. 2.6)
Church numerals, 93 (ex. 2.6)
Church-Turing thesis, 386 *n*
Chu Shih-chieh, 42 *n*
circuit
  digital (*see* digital-circuit simulation)
  modeled with streams, 344 (ex. 3.73), 349 (ex. 3.80)
Clark, Keith L., 466 *n*
clause, of a `cond`, 18
  additional syntax, 375 (ex. 4.5)
Clinger, William, 374 *n*, 402 *n*
closed world assumption, 466
closure, 82
  in abstract algebra, 98 *n*
  closure property of `cons`, 98
  closure property of picture-language operations, 126, 129
  lack of in many languages, 99 *n*
coal, bituminous, 128 *n*
code
  ASCII, 161
  fixed-length, 161
  Huffman (*see* Huffman code)
  Morse, 161
  prefix, 162
  variable-length, 161
code generator, 570
  arguments of, 571
  value of, 571

`coeff`, 205, *209*
coercion, 195–202
  in algebraic manipulation, 211
  in polynomial arithmetic, 207
  procedure, 195
  table, 195
Colmerauer, Alain, 439 *n*
combination, 5–7
  combination as operator of, 72 *n*
  compound expression as operator of, 21 (ex. 1.4)
  evaluation of, 9–11
  `lambda` expression as operator of, 63
  as operator of combination, 72 *n*
  as a tree, 9
combination, means of, 4. *See also* closure
comma, used with backquote, 575 *n*
comments in programs, 124 *n*
Common Lisp, 3 *n*
  treatment of `nil`, 101 *n*
compacting garbage collector, 541 *n*
compilation. *See* compiler
`compile`, *570*
`compile-and-go`, 604, *606*
`compile-and-run`, 609 (ex. 5.48)
`compile-application`, *582*
`compile-assignment`, *576*
`compile-definition`, *577*
`compile-if`, *578*
`compile-lambda`, *580*
`compile-linkage`, *575*
`compile-proc-appl`, *587*
`compile-procedure-call`, *584*
`compile-quoted`, *576*
`compile-self-evaluating`, *575*
`compile-sequence`, *579*
`compile-variable`, *576*
`compiled-apply`, *604*
`compiled-procedure?`, *580 n*
`compiled-procedure-entry`, *580 n*
`compiled-procedure-env`, *580 n*
compiler, 566–568
  interpreter vs., 567–568, 607
  tail recursion, stack allocation, and garbage-collection, 586 *n*
compiler for Scheme, 568–610. *See also* code generator; compile-time environment; instruction sequence; linkage descriptor; target register
  analyzing evaluator vs., 568, 569
  assignments, 576
  code generators (*see* `compile-...`)
  combinations, 581–587
  conditionals, 577
  definitions, 576
  efficiency, 568–569
  example compilation, 591–594
  explicit-control evaluator vs., 568–569, 574 (ex. 5.32), 606
  expression-syntax procedures, 570
  interfacing to evaluator, 603–610
  label generation, 578 *n*
  `lambda` expressions, 579
  lexical addressing, 600–602
  linkage code, 575
  machine-operation use, 567 *n*
  monitoring performance (stack use) of compiled code, 606, 608 (ex. 5.45), 609 (ex. 5.46)
  open coding of primitives, 595 (ex. 5.38), 603 (ex. 5.44)
  order of operand evaluation, 595 (ex. 5.36)
  procedure applications, 581–587
  quotations, 575
  register use, 567 *n*, 568, 587 *n*
  running compiled code, 603–610
  scanning out internal definitions, 602 *n*, 603 (ex. 5.43)
  self-evaluating expressions, 575
  sequences of expressions, 579
  stack usage, 572, 574 (ex. 5.31), 595 (ex. 5.37)
  structure of, 569–574
  tail-recursive code generated by, 586
  variables, 575
compile-time environment, 601, 602 (ex. 5.40), 602 (ex. 5.41)
  open coding and, 603 (ex. 5.44)

`complex` package, 191
`complex->complex`, *200* (ex. 2.81)
complex-number arithmetic, 170
  interfaced to generic arithmetic system, 191
  structure of system, 179 (fig. 2.21)
complex numbers
  polar representation, 174
  rectangular representation, 174
  rectangular vs. polar form, 171–172
  represented as tagged data, 175–179
composition of functions, 77 (ex. 1.42)
`compound-apply`, *554*
compound data, need for, 79–81
compound expression, 5. *See also* combination; special form
  as operator of combination, 21 (ex. 1.4)
compound procedure, 12. *See also* procedure
  used like primitive procedure, 13
`compound-procedure?`, *377*
compound query, 446–448
  processing, 456–458, 471–473, 488 (ex. 4.75), 488 (ex. 4.76), 489 (ex. 4.77)
computability, 386 *n*, 387 *n*
computational process, 1. *See also* process
computer science, 361, 386 *n*
  mathematics vs., 22, 438
concrete data representation, 83
concurrency, 297–316
  correctness of concurrent programs, 300–303
  deadlock, 314–315
  functional programming and, 356
  mechanisms for controlling, 303–316
`cond` (special form), *17*
  additional clause syntax, *375* (ex. 4.5)
  clause, 18
  evaluation of, 18
  `if` vs., 19 *n*
  implicit `begin` in consequent, 221 *n*
`cond?`, *373*
`cond-actions`, *373*
`cond-clauses`, *373*
`cond-else-clause?`, *373*
`cond->if`, *373*
`cond-predicate`, *373*
conditional expression
  `cond`, 17
  `if`, 19
congruent modulo *n*, 51
`conjoin`, *471*
`connect`, 289, *294*
connector(s), in constraint system, 286
  operations on, 289
  representing, 292
Conniver, 416 *n*
`cons` (primitive procedure), *85*
  axiom for, 91
  closure property of, 98
  implemented with mutators, 255
  implemented with vectors, 538
  as list operation, 100
  meaning of the name, 85 *n*
  procedural implementation of, *91*, *92* (ex. 2.4), *255*, *260*, *409*
`cons` up a list, 102
`cons-stream` (special form), 319, *321*
  lazy evaluation and, 409
  why a special form, 321 *n*
consciousness, expansion of, 367 *n*
consequent
  of `cond` clause, 18
  of `if`, 19
`const` (in register machine), 497
  simulating, 527
  syntax of, 513
constant, specifying in register machine, 513
`constant` (primitive constraint), *292*
`constant-exp`, *528*
`constant-exp-value`, *528*
constraint(s)
  primitive, 286
  propagation of, 285–296
constraint network, 286
`construct-arglist`, *583*

constructor, 83
  as abstraction barrier, 88
`contents`, *176*
  using Scheme data types, 193 (ex. 2.78)
continuation
  in nondeterministic evaluator, 427–428, 429 (*see also* failure continuation; success continuation)
  in register-machine simulator, 521 *n*
`continue` register, 504
  in explicit-control evaluator, 548
  recursion and, 508
continued fraction, 71 (ex. 1.37)
  *e* as, 71 (ex. 1.38)
  golden ratio as, 71 (ex. 1.37)
  tangent as, 72 (ex. 1.39)
controller for register machine, 492–494
  controller diagram, 494
control structure, 462
conventional interface, 82
  sequence as, 113–126
Cormen, Thomas H., 158 *n*
`corner-split`, *132*
correctness of a program, 22 *n*
`cos` (primitive procedure), *69*
cosine
  fixed point of, 69
  power series for, 332 (ex. 3.59)
cosmic radiation, 53 *n*
`count-change`, *40*
`count-leaves`, 108, 109
  as accumulation, 120 (ex. 2.35)
  as register machine, 539 (ex. 5.21)
`count-pairs`, *259* (ex. 3.16)
counting change, 40–41, 103 (ex. 2.19)
credit-card accounts, international, 316 *n*
Cressey, David, 542 *n*
cross-type operations, 194
cryptography, 53 *n*
`cube`, *44* (ex. 1.15), *56*, *74*
`cube-root`, *73*
cube root
  as fixed point, 73
  by Newton's method, 26 (ex. 1.8)
`current-time`, 280, *283*
current time, for simulation agenda, 283
cycle in list, 256 (ex. 3.13)
  detecting, 260 (ex. 3.18)

Darlington, John, 356 *n*
data, 1, 4
  abstract, 83 (*see also* data abstraction)
  abstract models for, 91 *n*
  algebraic specification for, 91 *n*
  compound, 79–81
  concrete representation of, 83
  hierarchical, 98, 107–111
  list-structured, 85
  meaning of, 90–93
  mutable (*see* mutable data objects)
  numerical, 5
  procedural representation of, 91–93
  as program, 384–387
  shared, 257–260
  symbolic, 142
  tagged, 175–179, 535 *n*
data abstraction, 80, 83, 169, 173, 368. *See also* metacircular evaluator
  for queue, 262
data base
  data-directed programming and, 185 (ex. 2.74)
  indexing, 455 *n*, 479
  Insatiable Enterprises personnel, 185 (ex. 2.74)
  logic programming and, 441
  Microshaft personnel, 441–443
  as set of records, 160
data-directed programming, 170, 179–186
  case analysis vs., 366
  in metacircular evaluator, 374 (ex. 4.3)
  in query interpreter, 470
data-directed recursion, 207
data paths for register machine, 492–494
  data-path diagram, 493

data types
  in Lisp, 193 (ex. 2.78)
  in strongly typed languages, 351 *n*
deadlock, 314–315
  avoidance, 314
  recovery, 314 *n*
debug, 2
decimal point in numbers, 24 *n*
declarative vs. imperative
    knowledge, 22, 438
  logic programming and, 439–440, 463
  nondeterministic computing and, 412 *n*
`decode`, *166*
decomposition of program into parts, 26
deep binding, 380 *n*
`deep-reverse`, 110 (ex. 2.27)
deferred operations, 34
`define` (special form), *7*
  with dotted-tail notation, 104 (ex. 2.20)
  environment model of, 240
  internal (*see* internal definition)
  `lambda` vs., 62–63
  for procedures, *12*, 62
  syntactic sugar, 370
  value of, 8 *n*
  why a special form, 11
`define-variable!`, 378, *380*
definite integral, 59–60
  estimated with Monte Carlo simulation, 228 (ex. 3.5), 354 (ex. 3.82)
definition. *See* `define`; internal definition
`definition?`, *370*
`definition-value`, *370*
`definition-variable`, *370*
deKleer, Johan, 416 *n*, 465 *n*
delay, in digital circuit, 273
`delay` (special form), *320*
  explicit, 347
  explicit vs. automatic, 411
  implementation using `lambda`, 323
  lazy evaluation and, 409
  memoized, 324, 332 (ex. 3.57)
  why a special form, 321 *n*
`delay-it`, *405*
delayed argument, 347
delayed evaluation, 218, 317
  assignment and, 325 (ex. 3.52)
  explicit vs. automatic, 411
  in lazy evaluator, 398–411
  normal-order evaluation and, 350–352
  printing and, 325 (ex. 3.51)
  streams and, 346–350
delayed object, 320
`delete-queue!`, 262, *265*
`denom`, 83, *86*
  axiom for, 90
  reducing to lowest terms, *89*
dense polynomial, 208
dependency-directed backtracking, 416 *n*
`deposit` message for bank account, *223*
`deposit`, with external serializer, *309*
depth-first search, 416
deque, 266 (ex. 3.23)
`deriv` (numerical), *74*
`deriv` (symbolic), *147*
  data-directed, *185* (ex. 2.73)
derivative of a function, 74
derived expressions in evaluator, 372–373
  adding to explicit-control evaluator, 560 (ex. 5.23)
design, stratified, 140
differential equation, 346. *See also* `solve`
  second-order, 348 (ex. 3.78), 349 (ex. 3.79)
differentiation
  numerical, 74
  rules for, 146, 150 (ex. 2.56)
  symbolic, 145–151, 184 (ex. 2.73)
diffusion, simulation of, 302
digital-circuit simulation, 273–285
  agenda, 280–281
  agenda implementation, 283–285
  primitive function boxes, 276–278
  representing wires, 278–280
  sample simulation, 281–282
digital signal, 273

Dijkstra, Edsger Wybe, 311 *n*
Dinesman, Howard P., 418
Diophantus's *Arithmetic*, Fermat's copy of, 51 *n*
`disjoin`, *471*
dispatching
  comparing different styles, 187 (ex. 2.76)
  on type, 179 (*see also* data-directed programming)
`display` (primitive procedure), 54 (ex. 1.22), *86 n*
`display-line`, *320*
`display-stream`, *320*
`distinct?`, *418 n*
`div` (generic), *189*
`div-complex`, *173*
`div-interval`, *94*
  division by zero, 95 (ex. 2.10)
`div-poly`, 210 (ex. 2.91)
`div-rat`, *84*
`div-series`, 334 (ex. 3.62)
`div-terms`, *210* (ex. 2.91)
`divides?`, *50*
`divisible?`, *326*
division of integers, 24 *n*
dog, perfectly rational, behavior of, 313 *n*
DOS/Windows, 565 *n*
`dot-product`, *121* (ex. 2.37)
dotted-tail notation
  for procedure parameters, 104 (ex. 2.20), 183 *n*
  in query-language rule, 451
  in query pattern, 445, 475
  `read` and, 475
Doyle, Jon, 416 *n*
`draw-line`, 136
`driver-loop`
  for lazy evaluator, *404*
  for metacircular evaluator, *383*
  for nondeterministic evaluator, *435*
driver loop
  in explicit-control evaluator, 560
  in lazy evaluator, 404
  in metacircular evaluator, 383
  in nondeterministic evaluator, 416, 434
  in query interpreter, 462, 468
*e*
  as continued fraction, 71 (ex. 1.38)
  as solution to differential equation, 348
$e^x$, power series for, 332 (ex. 3.59)
Earth, measuring circumference of, 327 *n*
`edge1-frame`, 134
`edge2-frame`, 134
efficiency. *See also* order of growth
  of compilation, 568
  of data-base access, 455 *n*
  of evaluation, 393
  of Lisp, 3
  of query processing, 457
  of tree-recursive process, 41
EIEIO, 315 *n*
eight-queens puzzle, 124 (ex. 2.42), 420 (ex. 4.44)
electrical circuits, modeled with streams, 344 (ex. 3.73), 349 (ex. 3.80)
`element-of-set?`, 152
  binary-tree representation, *157*
  ordered-list representation, *154*
  unordered-list representation, *152*
`else` (special symbol in `cond`), *18*
embedded language, language design using, 398
`empty-agenda?`, 280, *284*
`empty-arglist`, *551 n*
`empty-instruction-sequence`, *574*
`empty-queue?`, 262, *264*
`empty-termlist?`, 205, *209*
empty list, 101
  denoted as `'()`, 144
  recognizing with `null?`, 102
empty stream, 319
encapsulated name, 221 *n*
`enclosing-environment`, *378*
enclosing environment, 237
`encode`, *167* (ex. 2.68)
end-of-list marker, 99
`end-segment`, 89 (ex. 2.2), 137 (ex. 2.48)
`end-with-linkage`, *575*
engineering vs. mathematics, 53 *n*
`entry`, *156*

`enumerate-interval`, *116*
`enumerate-tree`, *116*
enumerator, 114
`env` register, 548
environment, 8, 236
  compile-time (*see* compile-time environment)
  as context for evaluation, 10
  enclosing, 237
  global (*see* global environment)
  lexical scoping and, 30 *n*
  in query interpreter, 490 (ex. 4.79)
  renaming vs., 489 (ex. 4.79)
environment model of evaluation, 218, 236–251
  environment structure, 237 (fig. 3.1)
  internal definitions, 249–251
  local state, 244–248
  message passing, 251 (ex. 3.11)
  metacircular evaluator and, 362
  procedure-application example, 241–244
  rules for evaluation, 238–241
  tail recursion and, 244 *n*
`eq?` (primitive procedure), *144*
  for arbitrary objects, 258
  as equality of pointers, 258, 535
  implementation for symbols, 536
  numerical equality and, 536 *n*
`equ?` (generic predicate), 193 (ex. 2.79)
`equal?`, 145 (ex. 2.54)
`equal-rat?`, *84*
equality
  in generic arithmetic system, 193 (ex. 2.79)
  of lists, 145 (ex. 2.54)
  of numbers, 18, 145 *n*, 536 *n*
  referential transparency and, 233
  of symbols, 144
equation, solving. *See* half-interval method; Newton's method; `solve`
Eratosthenes, 327 *n*
`error` (primitive procedure), *68 n*
error handling
  in compiled code, 607 *n*
  in explicit-control evaluator, 561, 565 (ex. 5.30)
Escher, Maurits Cornelis, 126 *n*
`estimate-integral`, 229 (ex. 3.5)
`estimate-pi`, *227*
Euclid's Algorithm, 48–49, 492. *See also* greatest common divisor
  order of growth, 49
  for polynomials, 212 *n*
Euclid's *Elements*, 48 *n*
Euclid's proof of infinite number of primes, 330 *n*
Euclidean ring, 212 *n*
Euler, Leonhard, 71 (ex. 1.38)
  proof of Fermat's Little Theorem, 51 *n*
  series accelerator, 336
`euler-transform`, *336*
`ev-application`, *551*
`ev-assignment`, *559*
`ev-begin`, *555*
`ev-definition`, *559*
`ev-if`, *558*
`ev-lambda`, *550*
`ev-quoted`, *550*
`ev-self-eval`, *550*
`ev-sequence`
  without tail recursion, *557*
  with tail recursion, *556*
`ev-variable`, *550*
`eval` (lazy), 402
`eval` (metacircular), 364, *365*
  analyzing version, *394*
  data-directed, *374* (ex. 4.3)
  primitive `eval` vs., 387 *n*
`eval` (primitive procedure), *387*
  MIT Scheme, 387 *n*
  used in query interpreter, 473
`eval-assignment`, *368*
`eval-definition`, *368*
`eval-dispatch`, *549*
`eval-if` (lazy), *403*
`eval-if` (metacircular), *367*
`eval-sequence`, *367*
evaluation
  applicative-order (*see* applicative-order evaluation)
  delayed (*see* delayed evaluation)
  environment model of (*see* environment model of evaluation)
  models of, 560

evaluation (*continued*)
normal-order (*see* normal-order evaluation)
of a combination, 9–11
of `and`, 19
of `cond`, 18
of `if`, 19
of `or`, 19
of primitive expressions, 10
of special forms, 11
order of subexpression evaluation (*see* order of evaluation)
substitution model of (*see* substitution model of procedure application)
evaluator, 360. *See also* interpreter
as abstract machine, 385
metacircular, 362
as universal machine, 386
evaluators. *See* metacircular evaluator; analyzing evaluator; lazy evaluator; nondeterministic evaluator; query interpreter; explicit-control evaluator
`even?`, *45*
`even-fibs`, *114*, *117*
evening star. *See* Venus
event-driven simulation, 273
evlis tail recursion, 552 *n*
exact integer, 24 *n*
`exchange`, *308*
exclamation point in names, 220 *n*
`execute`, 518
`execute-application`
metacircular, *397*
nondeterministic, *434*
execution procedure
in analyzing evaluator, 394
in nondeterministic evaluator, 426, 427, 429
in register-machine simulator, 517, 523–530
`exp` register, 548
`expand-clauses`, *373*
explicit-control evaluator for Scheme, 547–566
assignments, 559
combinations, 550–554
compound procedures, 554
conditionals, 558
controller, 549–562
data paths, 547–548
definitions, 559
derived expressions, 560 (ex. 5.23)
driver loop, 560
error handling, 561, 565 (ex. 5.30)
expressions with no subexpressions to evaluate, 549–550
as machine-language program, 566
machine model, 562
modified for compiled code, 603–605
monitoring performance (stack use), 563–565
normal-order evaluation, 560 (ex. 5.25)
operand evaluation, 551–553
operations, 547
optimizations (additional), 574 (ex. 5.32)
primitive procedures, 554
procedure application, 550–554
registers, 548
running, 560–563
sequences of expressions, 555–558
special forms (additional), 560 (ex. 5.23), 560 (ex. 5.24)
stack usage, 550
tail recursion, 556–558, 564 (ex. 5.26), 565 (ex. 5.28)
as universal machine, 566
`expmod`, *51*, *55* (ex. 1.25), *55* (ex. 1.26)
exponential growth, 43
of tree-recursive Fibonacci-number computation, 38
exponentiation, 44–46
modulo *n*, 51
expression. *See also* compound expression; primitive expression
algebraic (*see* algebraic expressions)
self-evaluating, 364
symbolic, 82 (*see also* symbol(s))
expression-oriented vs. imperative programming style, 296 *n*

`expt`
linear iterative version, *45*
linear recursive version, *44*
register machine for, 510 (ex. 5.4)
`extend-environment`, 378, *379*
`extend-if-consistent`, *474*
`extend-if-possible`, *479*
`external-entry`, *605*
`extract-labels`, *521*, *521 n*

`#f`, *18 n*
factorial, 32. *See also* `factorial`
infinite stream, 331 (ex. 3.54)
without `letrec` or `define`, 393 (ex. 4.21)
with `letrec`, 392 (ex. 4.20)
`factorial`
as an abstract machine, 384
compilation of, 591–594, 596 (fig. 5.17)
environment structure in evaluating, 243 (ex. 3.9)
linear iterative version, *34*
linear recursive version, *32*
register machine for (iterative), 494 (ex. 5.1), 498 (ex. 5.2)
register machine for (recursive), 506–509, 511 (fig. 5.11)
stack usage, compiled, 608 (ex. 5.45)
stack usage, interpreted, 564 (ex. 5.26), 564 (ex. 5.27)
stack usage, register machine, 532 (ex. 5.14)
with assignment, *235*
with higher-order procedures, 61 (ex. 1.31)
failure, in nondeterministic computation, 414
bug vs., 430
searching and, 415
failure continuation (nondeterministic evaluator), 427, 429
constructed by `amb`, 434
constructed by assignment, 432
constructed by driver loop, 434
false, 18 *n*
`false`, *18 n*
`false?`, *377*
`fast-expt`, *45*
`fast-prime?`, *52*
feedback loop, modeled with streams, 346
Feeley, Marc, 394 *n*
Feigenbaum, Edward, 440 *n*
Fenichel, Robert, 541 *n*
Fermat, Pierre de, 51 *n*
Fermat's Little Theorem, 51
alternate form, 56 (ex. 1.28)
proof, 51 *n*
`fermat-test`, *52*
Fermat test for primality, 51–53
variant of, 56 (ex. 1.28)
`fetch-assertions`, *480*
`fetch-rules`, *480*
`fib`
linear iterative version, *39*
logarithmic version, 47 (ex. 1.19)
register machine for (tree-recursive), 510, 512 (fig. 5.12)
stack usage, compiled, 609 (ex. 5.46)
stack usage, interpreted, 565 (ex. 5.29)
tree-recursive version, *37*, *565* (ex. 5.29)
with memoization, *272* (ex. 3.27)
with named `let`, *376* (ex. 4.8)
Fibonacci numbers, 37. *See also* `fib`
Euclid's GCD algorithm and, 49
infinite stream of (*see* `fibs`)
`fibs` (infinite stream), *327*
implicit definition, *329*
FIFO buffer, 261
filter, 61 (ex. 1.33), 114
`filter`, *115*
`filtered-accumulate`, 61 (ex. 1.33)
`find-assertions`, *473*
`find-divisor`, *50*
`first-agenda-item`, 280, *285*
`first-exp`, *371*
`first-frame`, *378*

`first-operand`, *372*
`first-segment`, *283*
`first-term`, 205, *209*
first-class elements in language, 76
fixed-length code, 161
fixed point, 68–70
  computing with calculator, 69 *n*
  of cosine, 69
  cube root as, 73
  fourth root as, 78 (ex. 1.45)
  golden ratio as, 70 (ex. 1.35)
  as iterative improvement, 78 (ex. 1.46)
  in Newton's method, 73
  *n*th root as, 78 (ex. 1.45)
  square root as, 69, 73, 75
  of transformed function, 75
  unification and, 478 *n*
`fixed-point`, *69*
  as iterative improvement, 78 (ex. 1.46)
`fixed-point-of-transform`, *75*
`flag` register, 517
`flatmap`, *123*
`flatten-stream`, *483*
`flip-horiz`, 128, 140 (ex. 2.50)
`flip-vert`, 128, *138*
`flipped-pairs`, *130*, *133 n*, *134*
Floyd, Robert, 416 *n*
`fold-left`, *121* (ex. 2.38)
`fold-right`, 121 (ex. 2.38)
`for-each`, 107 (ex. 2.23), *407* (ex. 4.30)
`for-each-except`, *294*
Forbus, Kenneth D., 416 *n*
`force`, 320, *323*
  forcing a thunk vs., 401 *n*
force a thunk, 401
`force-it`, *405*
  memoized version, *406*
`forget-value!`, 289, *294*
formal parameters, 12
  names of, 28
  scope of, 29
formatting input expressions, 7 *n*
Fortran, 3, 118 *n*
  inventor of, 356 *n*
  restrictions on compound data, 99 *n*
forwarding address, 542
fourth root, as fixed point, 78 (ex. 1.45)
fraction. *See* rational number(s)
frame (environment model), 237
  as repository of local state, 244–248
  global, 237
frame (picture language), 127, 134
  coordinate map, 134
frame (query interpreter), 454. *See also* pattern matching; unification
  representation, 486
`frame-coord-map`, *135*
`frame-values`, *378*
`frame-variables`, *378*
framed-stack discipline, 550 *n*
Franz Lisp, 3 *n*
`free` register, 538, 542
free list, 538 *n*
free variable, 28
  capturing, 29
  in internal definition, 30
Friedman, Daniel P., 324 *n*, 361 *n*
`fringe`, 111 (ex. 2.28)
  as a tree enumeration, 116 *n*
`front-ptr`, *263*
`front-queue`, 262, *264*
full-adder, 275
  `full-adder`, *276*
function (mathematical)
  $\mapsto$ notation for, 69 *n*
  Ackermann's, 36 (ex. 1.10)
  composition of, 77 (ex. 1.42)
  derivative of, 74
  fixed point of, 68–70
  procedure vs., 21–22
  rational, 211–216
  repeated application of, 77 (ex. 1.43)
  smoothing of, 78 (ex. 1.44)
functional programming, 230, 352–357
  concurrency and, 356
  functional programming languages, 356
  time and, 354–357
function box, in digital circuit, 273

Gabriel, Richard P., 393 *n*
garbage collection, 540–546
  memoization and, 405 *n*
  mutation and, 253 *n*
  tail recursion and, 586 *n*
garbage collector
  compacting, 541 *n*
  mark-sweep, 541 *n*
  stop-and-copy, 540–546
GCD. *See* greatest common divisor
`gcd`, *49*
  register machine for, 492–494, 514
`gcd-terms`, *213*
general-purpose computer, as universal machine, 566
`generate-huffman-tree`, *168* (ex. 2.69)
generating sentences, 426 (ex. 4.49)
generic arithmetic operations, 189–193
  structure of system, 188 (fig. 2.23)
generic operation, 82
generic procedure, 166, 170
  generic selector, 177, 179
Genesis, 453 (ex. 4.63)
`get`, 181, *271*
`get-contents`, *516*
`get-global-environment`, *561 n*
`get-register`, *518*
`get-register-contents`, 514, *518*
`get-signal`, 276, *280*
`get-value`, 289, *294*
glitch, 1
global environment, 8, 238
  in metacircular evaluator, 381
global frame, 237
Goguen, Joseph, 91 *n*
golden ratio, 38
  as continued fraction, 71 (ex. 1.37)
  as fixed point, 70 (ex. 1.35)
Gordon, Michael, 351 *n*
`goto` (in register machine), 496
  label as destination, 506
  simulating, 526
`goto-dest`, *526*
grammar, 421
graphics. *See* picture language
Gray, Jim, 314 *n*
greatest common divisor, 48–49. *See also* `gcd`
  generic, 213 (ex. 2.94)
  of polynomials, 212
  used to estimate $\pi$, 226
  used in rational-number arithmetic, 87
Green, Cordell, 439 *n*
Griss, Martin Lewis, 3 *n*
Guttag, John Vogel, 91 *n*

half-adder, 274
  `half-adder`, *275*
  simulation of, 281–282
half-interval method, 67–68
  `half-interval-method`, *68*
  Newton's method vs., 74 *n*
halting problem, 387 (ex. 4.15)
Halting Theorem, 387 *n*
Hamming, Richard Wesley, 164 *n*, 331 (ex. 3.56)
Hanson, Christopher P., 374 *n*, 586 *n*
Hardy, Godfrey Harold, 330 *n*, 342 *n*
`has-value?`, 289, *294*
Hassle, 400 *n*
Havender, J., 314 *n*
Haynes, Christopher T., 361 *n*
headed list, 267, 283 *n*
Hearn, Anthony C., 3 *n*
Henderson, Peter, 126 *n*, 328 *n*, 356 *n*
  Henderson diagram, 328
Heraclitus, 217
Heron of Alexandria, 23 *n*
Hewitt, Carl Eddie, 36 *n*, 416 *n*, 439 *n*, 541 *n*
hiding principle, 221 *n*
hierarchical data structures, 98, 107–111
hierarchy of types, 197–202
  inadequacy of, 198
  in symbolic algebra, 210–211
higher-order procedures, 57
  in metacircular evaluator, 366 *n*
  procedure as argument, 57–61
  procedure as general method, 66–72
  procedure as returned value, 72–78
  strong typing and, 351 *n*

high-level language, machine language vs., 360
Hilfinger, Paul, 160 *n*
Hoare, Charles Antony Richard, 91 *n*
Hodges, Andrew, 386 *n*
Hofstadter, Douglas R., 386 *n*
Horner, W. G., 119 *n*
Horner's rule, 119 (ex. 2.34)
"how to" vs. "what is" description. *See* imperative vs. declarative knowledge
Huffman, David, 162
Huffman code, 161–169
  optimality of, 164
  order of growth of encoding, 169 (ex. 2.72)
Hughes, R. J. M., 410 *n*

IBM 704, 85 *n*
`identity`, *59*
`if` (special form), *19*
  `cond` vs., 19 *n*
  evaluation of, 19
  normal-order evaluation of, 21 (ex. 1.5)
  one-armed (without alternative), 284 *n*
  predicate, consequent, and alternative of, 19
  why a special form, 25 (ex. 1.6)
`if?`, *371*
`if-alternative`, *371*
`if-consequent`, *371*
`if-predicate`, *371*
`imag-part`
  data-directed, *184*
  polar representation, *175*
  rectangular representation, *174*
  with tagged data, *177*
`imag-part-polar`, *177*
`imag-part-rectangular`, *176*
imperative programming, 234
imperative vs. declarative knowledge, 22, 438
  logic programming and, 439–440, 463
  nondeterministic computing and, 412 *n*
imperative vs. expression-oriented programming style, 296 *n*
implementation dependencies. *See also* unspecified values
  numbers, 24 *n*
  order of subexpression evaluation, 238 *n*
`inc`, *58*
incremental development of programs, 8
indeterminate of a polynomial, 202
indexing a data base, 455 *n*, 479
inference, method of, 462
infinite series, 478 *n*
infinite stream(s), 326–334
  merging, 331 (ex. 3.56), 340, 342 (ex. 3.70), 356
  merging as a relation, 357 *n*
  of factorials, 331 (ex. 3.54)
  of Fibonacci numbers (*see* `fibs`)
  of integers (*see* `integers`)
  of pairs, 338–343
  of prime numbers (*see* `primes`)
  of random numbers, 352
  representing power series, 332 (ex. 3.59)
  to model signals, 343–346
  to sum a series, 335
infix notation, prefix notation vs., 151 (ex. 2.58)
`inform-about-no-value`, *290*
`inform-about-value`, *290*
information retrieval. *See* data base
Ingerman, Peter, 401 *n*
`initialize-stack` operation in register machine, 517, 530
`insert!`
  in one-dimensional table, *268*
  in two-dimensional table, *270*
`insert-queue!`, 262, *264*
`install-complex-package`, *191*
`install-polar-package`, *183*
`install-polynomial-package`, *204*
`install-rational-package`, *190*
`install-rectangular-package`, *182*

`install-scheme-number-package`, *189*
`instantiate`, *470*
instantiate a pattern, 445
`instruction-execution-proc`, *522*
`instruction-text`, *522*
instruction counting, 532 (ex. 5.15)
instruction execution procedure, 517
instruction sequence, 571–574, 587–591
instruction tracing, 532 (ex. 5.16)
integer(s), 5 *n*
  dividing, 24 *n*
  exact, 24 *n*
integerizing factor, 214
`integers` (infinite stream), *326*
  implicit definition, *329*
  lazy-list version, *410*
`integers-starting-from`, *326*
integral. *See also* definite integral; Monte Carlo integration
  of a power series, 332 (ex. 3.59)
`integral`, *60*, *343*, *348* (ex. 3.77)
  with delayed argument, *347*
  with `lambda`, *62*
  lazy-list version, *411*
  need for delayed evaluation, 346
`integrate-series`, 332 (ex. 3.59)
integrated-circuit implementation of Scheme, 547, 548 (fig. 5.16)
integrator, for signals, 343
`interleave`, *341*
`interleave-delayed`, *483*
Interlisp, 3 *n*
internal definition, 30–31
  in environment model, 249–251
  free variable in, 30
  `let` vs., 66
  in nondeterministic evaluator, 432 *n*
  position of, 31 *n*
  restrictions on, 388
  scanning out, 389
  scope of name, 388–390
Internet "Worm", 607 *n*
interning symbols, 537
interpreter, 2. *See also* evaluator
  compiler vs., 567–568, 607
  read-eval-print loop, 7
`intersection-set`, 152
  binary-tree representation, 160 (ex. 2.65)
  ordered-list representation, *155*
  unordered-list representation, *153*
interval arithmetic, 93–97
invariant quantity of an iterative process, 46 (ex. 1.16)
inverter, 273
  `inverter`, *277*
iteration contructs. *See* looping constructs
iterative improvement, 78 (ex. 1.46)
iterative process, 34
  as a stream process, 334–338
  design of algorithm, 46 (ex. 1.16)
  implemented by procedure call, 24–25, 36, 558 (*see also* tail recursion)
  linear, 34, 43
  recursive process vs., 32–36, 243 (ex. 3.9), 506, 594 (ex. 5.34)
  register machine for, 506

Jayaraman, Sundaresan, 286 *n*

Kaldewaij, Anne, 47 *n*
Karr, Alphonse, 217
Kepler, Johannes, 491
`key`, 160
key of a record
  in a data base, 160
  in a table, 266
  testing equality of, 272 (ex. 3.24)
Khayyam, Omar, 42 *n*
Knuth, Donald E., 42 *n*, 46 *n*, 48 *n*, 119 *n*, 226 *n*, 621
Kohlbecker, Eugene Edmund, Jr., 374 *n*
Kolmogorov, A. N., 226 *n*
Konopasek, Milos, 286 *n*
Kowalski, Robert, 439 *n*
KRC, 122 *n*, 340 *n*

`label` (in register machine), 496
  simulating, 527
`label-exp`, *528*
`label-exp-label`, *528*

Lagrange interpolation formula, 203 *n*
λ calculus (lambda calculus), 63 *n*
`lambda` expression
  as operator of combination, 63
  value of, 240
`lambda` (special form), *62*
  `define` vs., 62–63
  with dotted-tail notation, 104 *n*
`lambda?`, *370*
`lambda-body`, *370*
`lambda-parameters`, *370*
Lambert, J.H., 72 (ex. 1.39)
Lamé, Gabriel, 49 *n*
Lamé's Theorem, 49
Lamport, Leslie, 316 *n*
Lampson, Butler, 234 *n*
Landin, Peter, 11 *n*, 324 *n*
language. *See* natural language; programming language
Lapalme, Guy, 394 *n*
`last-exp?`, *371*
`last-operand?`, *551 n*
`last-pair`, 103 (ex. 2.17), *255* (ex. 3.12)
  rules, 453 (ex. 4.62)
lazy evaluation, 399
lazy evaluator, 398–409
lazy list, 409–411
lazy pair, 409–411
lazy tree, 410 *n*
`leaf?`, *165*
least commitment, principle of, 175
lecture, something to do during, 69 *n*
`left-branch`, *156*, *165*
Leibniz, Baron Gottfried Wilhelm von
  proof of Fermat's Little Theorem, 51 *n*
  series for $\pi$, 57 *n*, 335
Leiserson, Charles E., 158 *n*, 342 *n*
`length`, 102
  as accumulation, 119 (ex. 2.33)
  iterative version, *102*
  recursive version, *102*
`let` (special form), *64*
  evaluation model, 248 (ex. 3.10)
  internal definition vs., 66
  named, *376* (ex. 4.8)
  scope of variables, 65
  as syntactic sugar, 65, 248 (ex. 3.10)
`let*` (special form), *375* (ex. 4.7)
`letrec` (special form), *391* (ex. 4.20)
`lexical-address-lookup`, 601, 602 (ex. 5.39)
`lexical-address-set!`, 601, 602 (ex. 5.39)
lexical addressing, 600–602
  lexical address, 600
lexical scoping, 30
  environment structure and, 600
Lieberman, Henry, 541 *n*
LIFO buffer. *See* stack
linear growth, 34, 43
linear iterative process, 34
  order of growth, 43
linear recursive process, 34
  order of growth, 43
line segment
  represented as pair of points, 89 (ex. 2.2)
  represented as pair of vectors, 137 (ex. 2.48)
linkage descriptor, 571
Liskov, Barbara Huberman, 91 *n*
Lisp
  acronym for LISt Processing, 2
  applicative-order evaluation in, 17
  on DEC PDP-1, 541 *n*
  efficiency of, 3, 7 *n*
  first-class procedures in, 77
  Fortran vs., 3
  history of, 2–4
  internal type system, 193 (ex. 2.78)
  original implementation on IBM 704, 85 *n*
  Pascal vs., 11 *n*
  suitability for writing evaluators, 361
  unique features of, 4
`lisp-value` (query interpreter), *472*
`lisp-value` (query language), 447, 466
  evaluation of, 458, 472, 489 (ex. 4.77)

Lisp dialects
  Common Lisp, 3 *n*
  Franz Lisp, 3 *n*
  Interlisp, 3 *n*
  MacLisp, 3 *n*
  MDL, 542 *n*
  Portable Standard Lisp, 3 *n*
  Scheme, 3
  Zetalisp, 3 *n*
list(s), 100
  backquote with, 575 *n*
  `cdr`ing down, 101
  combining with `append`, 102
  `cons`ing up, 102
  converting a binary tree to a, 158 (ex. 2.63)
  converting to a binary tree, 159 (ex. 2.64)
  empty (*see* empty list)
  equality of, 145 (ex. 2.54)
  headed, 267, 283 *n*
  last pair of, 103 (ex. 2.17)
  lazy, 409–411
  length of, 102
  list structure vs., 100 *n*
  manipulation with `car`, `cdr`, and `cons`, 100
  mapping over, 105–107
  *n*th element of, 101
  operations on, 101–104
  printed representation of, 100
  quotation of, 143
  reversing, 103 (ex. 2.18)
  techniques for manipulating, 101–104
`list` (primitive procedure), *100*
`list-difference`, *589*
`list-of-arg-values`, *403*
`list-of-delayed-args`, *403*
`list-of-values`, *367*
`list-ref`, *101*, *410*
`list->tree`, *159* (ex. 2.64)
`list-union`, *589*
list structure, 85
  list vs., 100 *n*
  mutable, 252–256
  represented using vectors, 535–539
list-structured memory, 533–546
`lives-near` (rule), *448*, 451 (ex. 4.60)
local evolution of a process, 31
local name, 27–29, 63–66
local state, 218–236
  maintained in frames, 244–248
local state variable, 219–225
local variable, 63–66
location, 534
Locke, John, 1
`log` (primitive procedure), *70* (ex. 1.36)
logarithm, approximating ln 2, 338 (ex. 3.65)
logarithmic growth, 43, 45, 156 *n*
logical and, 273
`logical-not`, *277*
logical or, 274
logic programming, 438–441. *See also* query language; query interpreter
  computers for, 440 *n*
  history of, 438 *n*, 440 *n*
  in Japan, 440 *n*
  logic programming languages, 440
  mathematical logic vs., 462–468
logic puzzles, 418–420
`lookup`
  in one-dimensional table, *268*
  in set of records, *160*
  in two-dimensional table, *270*
`lookup-label`, *522*
`lookup-prim`, *529*
`lookup-variable-value`, 377, *379*
  for scanned-out definitions, 390 (ex. 4.16)
looping constructs, 25, 35
  implementing in metacircular evaluator, 376 (ex. 4.9)
`lower-bound`, 94 (ex. 2.7)

machine language, 566
  high-level language vs., 360
Macintosh, 565 *n*
MacLisp, 3 *n*
macro, 373 *n*. *See also* reader macro character
magician. *See* numerical analyst

magnitude
  data-directed, *184*
  polar representation, *175*
  rectangular representation, *174*
  with tagged data, *178*
magnitude-polar, *177*
magnitude-rectangular, *176*
make-account, *223*
  in environment model, 251 (ex. 3.11)
  with serialization, *305*, *306* (ex. 3.41), *307* (ex. 3.42)
make-account-and-serializer, *309*
make-accumulator, 224 (ex. 3.1)
make-agenda, 280, *283*
make-assign, *524*
make-begin, *372*
make-branch, *526*
make-center-percent, 96 (ex. 2.12)
make-center-width, *95*
make-code-tree, *165*
make-compiled-procedure, *580n*
make-complex-from-mag-ang, *192*
make-complex-from-real-imag, *192*
make-connector, *293*
make-cycle, *256* (ex. 3.13)
make-decrementer, *230*
make-execution-procedure, *523*
make-frame, 134, *136* (ex. 2.47), *378*
make-from-mag-ang, *178*, *184*
  message-passing, *187* (ex. 2.75)
  polar representation, *175*
  rectangular representation, *174*
make-from-mag-ang-polar, *177*
make-from-mag-ang-rectangular, *177*
make-from-real-imag, *178*, *184*
  message-passing, *186*
  polar representation, *175*
  rectangular representation, *174*
make-from-real-imag-polar, *177*
make-from-real-imag-rectangular, *177*
make-goto, *526*
make-if, *371*
make-instruction, *522*
make-instruction-sequence, *573*
make-interval, 94, *94* (ex. 2.7)
make-joint, 236 (ex. 3.7)
make-label, *578n*
make-label-entry, *522*
make-lambda, *370*
make-leaf, *165*
make-leaf-set, *167*
make-machine, 514, *516*
make-monitored, 224 (ex. 3.2)
make-mutex, *312*
make-new-machine, *519* (fig. 5.12)
make-operation-exp, *528*
make-perform, *527*
make-point, 89 (ex. 2.2)
make-poly, *204*
make-polynomial, *209*
make-primitive-exp, *527*
make-procedure, *377*
make-product, *148*, *150*
make-queue, 262, *264*
make-rat, 83, *86*, *89*
  axiom for, 90
  reducing to lowest terms, *87*
make-rational, *190*
make-register, *516*
make-restore, *527*
make-save, *526*
make-scheme-number, *189*
make-segment, 89 (ex. 2.2), 137 (ex. 2.48)
make-serializer, *311*
make-simplified-withdraw, *230*, *354*
make-stack, *517*
  with monitored stack, *531*
make-sum, *148*, *149*
make-table
  message-passing implementation, *271*
  one-dimensional table, *268*
make-tableau, *337*
make-term, 205, *209*
make-test, *525*
make-time-segment, *283*

`make-tree`, *157*
`make-vect`, 136 (ex. 2.46)
`make-wire`, 274, *279*, 282 (ex. 3.31)
`make-withdraw`, *222*
  in environment model, 244–248
  using `let`, *248* (ex. 3.10)
making change. *See* counting change
`map`, *105*, *410*
  as accumulation, 119 (ex. 2.33)
  with multiple arguments, 105 *n*
`map-over-symbols`, *485*
`map-successive-pairs`, *353*
mapping
  over lists, 105–107
  nested, 122–126, 338–343
  as a transducer, 114
  over trees, 112–113
mark-sweep garbage collector, 541 *n*
mathematical function. *See* function (mathematical)
mathematics
  computer science vs., 22, 438
  engineering vs., 53 *n*
matrix, represented as sequence, 120 (ex. 2.37)
`matrix-*-matrix`, 121 (ex. 2.37)
`matrix-*-vector`, 121 (ex. 2.37)
`max` (primitive procedure), *94*
McAllester, David Allen, 416 *n*
McCarthy, John, 2, 2 *n*, 414 *n*
McDermott, Drew, 416 *n*
MDL, 542 *n*
means of abstraction, 4
  `define`, 8
means of combination, 4. *See also* closure
measure in a Euclidean ring, 212 *n*
`member`, 418 *n*
`memo-fib`, *272* (ex. 3.27)
`memo-proc`, *324*
memoization, 41 *n*, 272 (ex. 3.27)
  call-by-need and, 332 *n*
  by `delay`, 324
  garbage collection and, 405 *n*
  of thunks, 401
`memoize`, *273* (ex. 3.27)
memory
  in 1964, 415 *n*
  list-structured, 533–546
`memq`, *144*
`merge`, *331* (ex. 3.56)
`merge-weighted`, 342 (ex. 3.70)
merging infinite streams. *See* infinite stream(s)
message passing, 92, 186–188
  environment model and, 251 (ex. 3.11)
  in bank account, 223
  in digital-circuit simulation, 278
  tail recursion and, 36 *n*
metacircular evaluator, 362
metacircular evaluator for Scheme, 362–387
  analyzing version, 393–398
  combinations (procedure applications), 374 (ex. 4.2)
  compilation of, 610 (ex. 5.50), 610 (ex. 5.52)
  data abstraction in, 363, 364, 376 (ex. 4.10), 380
  data-directed `eval`, 374 (ex. 4.3)
  derived expressions, 372–373
  driver loop, 383
  efficiency of, 393
  environment model of evaluation in, 362
  environment operations, 377
  `eval` and `apply`, 364–368
  `eval`–`apply` cycle, 363, 364 (fig. 4.1)
  expression representation, 364, 368–373
  global environment, 381
  higher-order procedures in, 366 *n*
  implemented language vs. implementation language, 367
  job of, 363 *n*
  order of operand evaluation, 368 (ex. 4.1)
  primitive procedures, 381–383
  representation of environments, 378–380

metacircular evaluator for Scheme (*continued*)
representation of procedures, 377
representation of true and false, 376
running, 381–384
special forms (additional), 374 (ex. 4.4), 375 (ex. 4.5), 375 (ex. 4.6), 375 (ex. 4.7), 376 (ex. 4.8), 376 (ex. 4.9)
special forms as derived expressions, 372–373
symbolic differentiation and, 368
syntax of evaluated language, 368–373, 374 (ex. 4.2), 376 (ex. 4.10)
tail recursiveness unspecified in, 556
`true` and `false`, 381
metalinguistic abstraction, 360
MicroPlanner, 416 *n*
Microshaft, 441
`midpoint-segment`, 90 (ex. 2.2)
Miller, Gary L., 56 (ex. 1.28)
Miller, James S., 586 *n*
Miller-Rabin test for primality, 56 (ex. 1.28)
Milner, Robin, 351 *n*
`min` (primitive procedure), *94*
Minsky, Marvin Lee, xvii, 541 *n*
Miranda, 122 *n*
MIT, 439 *n*
Artificial Intelligence Laboratory, 3 *n*
early history of, 127 *n*
Project MAC, 3 *n*
Research Laboratory of Electronics, 2, 541 *n*
MIT Scheme
the empty stream, 319 *n*
`eval`, 387 *n*
internal definitions, 390 *n*
numbers, 24 *n*
`random`, 229 *n*
`user-initial-environment`, 387 *n*
`without-interrupts`, 313 *n*
ML, 351 *n*
mobile, 111 (ex. 2.29)
modeling
as a design strategy, 217
in science and engineering, 15
models of evaluation, 560
modified registers. *See* instruction sequence
`modifies-register?`, *588*
modularity, 117, 217
along object boundaries, 357 *n*
functional programs vs. objects, 352–357
hiding principle, 221 *n*
streams and, 334
through dispatching on type, 179
through infinite streams, 353
through modeling with objects, 225
modulo *n*, 51
*modus ponens*, 462 *n*
money, changing. *See* counting change
monitored procedure, 224 (ex. 3.2)
`monte-carlo`, *227*
infinite stream, *353*
Monte Carlo integration, 228 (ex. 3.5)
stream formulation, 354 (ex. 3.82)
Monte Carlo simulation, 226
stream formulation, 352
Moon, David A., 3 *n*, 541 *n*
morning star. *See* evening star
Morris, J. H., 234 *n*
Morse code, 161
Mouse, Minnie and Mickey, 464
`mul` (generic), *189*
used for polynomial coefficients, 206
`mul-complex`, *173*
`mul-interval`, *94*
more efficient version, 95 (ex. 2.11)
`mul-poly`, *204*
`mul-rat`, *84*
`mul-series`, 333 (ex. 3.60)
`mul-streams`, 331 (ex. 3.54)
`mul-terms`, *206*
Multics time-sharing system, 541 *n*
`multiple-dwelling`, *419*
`multiplicand`, *149*

multiplication by Russian peasant method, 47 *n*
`multiplier`
  primitive constraint, *291*
  selector, *148*
Munro, Ian, 119 *n*
mutable data objects, 251–261. *See also* queue; table
  implemented with assignment, 260–261
  list structure, 252–256
  pairs, 252–256
  procedural representation of, 260–261
  shared data, 258
mutator, 252
mutex, 311
mutual exclusion, 311 *n*
`mystery`, *256* (ex. 3.14)

name. *See also* local name; variable; local variable
  encapsulated, 221 *n*
  of a formal parameter, 28
  of a procedure, 12
named `let` (special form), *376* (ex. 4.8)
naming
  of computational objects, 7
  of procedures, 12
naming conventions
  `!` for assignment and mutation, 220 *n*
  `?` for predicates, 24 *n*
native language of machine, 566
natural language
  parsing (*see* parsing natural language)
  quotation in, 142
needed registers. *See* instruction sequence
`needs-register?`, *588*
`negate`, *472*
nested applications of `car` and `cdr`, 100 *n*
nested combinations, 6–7
nested definitions. *See* internal definition
nested mappings. *See* mapping
`new` register, 544
`new-cars` register, 542
`new-cdrs` register, 542
`new-withdraw`, *221*
`newline` (primitive procedure), 54 (ex. 1.22), *86 n*
Newton's method
  for cube roots, 26 (ex. 1.8)
  for differentiable functions, 73–75
  half-interval method vs., 74 *n*
  for square roots, 22–24, 75, 76
`newton-transform`, *74*
`newtons-method`, *75*
`next` (linkage descriptor), 571
`next-to` (rules), *452* (ex. 4.61)
`nil`
  dispensing with, 144
  as empty list, 101
  as end-of-list marker, 99
  as ordinary variable in Scheme, 101 *n*
`no-more-exps?`, *557 n*
`no-operands?`, *372*
node of a tree, 9
non-computable, 387 *n*
nondeterminism, in behavior of concurrent programs, 302 *n*, 357 *n*
nondeterministic choice point, 415
nondeterministic computing, 412–426
nondeterministic evaluator, 426–437
  order of operand evaluation, 425 (ex. 4.46)
nondeterministic programming vs. Scheme programming, 412, 420 (ex. 4.41), 420 (ex. 4.44), 489 (ex. 4.78)
nondeterministic programs
  logic puzzles, 418–419
  pairs with prime sums, 412
  parsing natural language, 420–425
  Pythagorean triples, 417 (ex. 4.35), 417 (ex. 4.36), 418 (ex. 4.37)
non-strict, 400
normal-order evaluation, 16
  applicative order vs., 21 (ex. 1.5), 49 (ex. 1.20), 399–401

normal-order evaluation (*continued*)
delayed evaluation and, 350–352
in explicit-control evaluator, 560 (ex. 5.25)
of `if`, 21 (ex. 1.5)
normal-order evaluator. *See* lazy evaluator
`not` (primitive procedure), *19*
`not` (query language), 447, 465
evaluation of, 457, 472, 489 (ex. 4.77)
notation in this book
italic symbols in expression syntax, 12 *n*
slanted characters for interpreter response, 5 *n*
`nouns`, *421*
*n*th root, as fixed point, 78 (ex. 1.45)
`null?` (primitive procedure), *102*
implemented with typed pointers, 538
number(s)
comparison of, 18
decimal point in, 24 *n*
equality of, 18, 145 *n*, 536 *n*
in generic arithmetic system, 189
implementation dependencies, 24 *n*
integer, exact, 24 *n*
integer vs. real number, 5 *n*
in Lisp, 5
rational number, 24 *n*
`number?` (primitive procedure), *147*
data types and, 193 (ex. 2.78)
implemented with typed pointers, 538
number theory, 51 *n*
`numer`, 83, *86*
axiom for, 90
reducing to lowest terms, *89*
numerical analysis, 5 *n*
numerical analyst, 67 *n*
numerical data, 5

obarray, 537
object(s), 218
benefits of modeling with, 225
with time-varying state, 219
object-oriented programming languages, 200 *n*
object program, 567
`old` register, 544
`oldcr` register, 545
`ones` (infinite stream), *328*
lazy-list version, *410*
`op` (in register machine), 497
simulating, 528
open coding of primitives, 595 (ex. 5.38), 603 (ex. 5.44)
`operands`, *372*
operands of a combination, 6
operation
cross-type, 194
generic, 82
in register machine, 492–494
operation-and-type table, 181
assignment needed for, 220 *n*
implementing, 271
`operation-exp`, *528*
`operation-exp-op`, *528*
`operation-exp-operands`, *528*
`operator`, *372*
operator of a combination, 6
combination as, 72 *n*
compound expression as, 21 (ex. 1.4)
`lambda` expression as, 63
optimality
of Horner's rule, 119 *n*
of Huffman code, 164
`or` (query language), 446
evaluation of, 456, 471
`or` (special form), *19*
evaluation of, 19
why a special form, 19
with no subexpressions, *374* (ex. 4.4)
`order`, 205, *209*
ordered-list representation of sets, 153–155
order notation, 43
order of evaluation
assignment and, 236 (ex. 3.8)
implementation-dependent, 238 *n*
in compiler, 595 (ex. 5.36)
in explicit-control evaluator, 553
in metacircular evaluator, 368 (ex. 4.1)
in Scheme, 236 (ex. 3.8)

order of events
  decoupling apparent from actual, 323
  indeterminacy in concurrent systems, 298
order of growth, 42–43
  linear iterative process, 43
  linear recursive process, 43
  logarithmic, 45
  tree-recursive process, 43
order of subexpression evaluation. *See* order of evaluation
ordinary numbers (in generic arithmetic system), 189
or-gate, 274
  `or-gate`, 277 (ex. 3.28), 278 (ex. 3.29)
`origin-frame`, 134
Ostrowski, A. M., 119 *n*
`outranked-by` (rule), *449*, 466 (ex. 4.64)

P operation on semaphore, 311 *n*
package, 182
  complex-number, 191
  polar representation, 183
  polynomial, 204
  rational-number, 190
  rectangular representation, 182
  Scheme-number, 189
painter(s), 127
  higher-order operations, 132
  operations, 128
  represented as procedures, 136
  transforming and combining, 138
pair(s), 85
  axiomatic definition of, 91
  box-and-pointer notation for, 97
  infinite stream of, 338–343
  lazy, 409–411
  mutable, 252–256
  procedural representation of, 91–92, 260–261, 409
  represented using vectors, 535–539
  used to represent sequence, 99
  used to represent tree, 107–111
`pair?` (primitive procedure), *109*
  implemented with typed pointers, 538
`pairs`, *341*
Pan, V. Y., 119 *n*
`parallel-execute`, 304
`parallel-instruction-sequences`, *591*
parallelism. *See* concurrency
parameter. *See* formal parameters
parameter passing. *See* call-by-name argument passing; call-by-need argument passing
parentheses
  delimiting combination, 6
  delimiting `cond` clauses, 18
  in procedure definition, 13
`parse`, *422*
`parse-...`, 421–423
parsing natural language, 420–426
  real language understanding vs. toy parser, 426 *n*
`partial-sums`, 331 (ex. 3.55)
Pascal, 11 *n*
  lack of higher-order procedures, 351 *n*
  recursive procedures, 35
  restrictions on compound data, 99 *n*
  weakness in handling compound objects, 296 *n*
Pascal, Blaise, 42 *n*
Pascal's triangle, 42 (ex. 1.12)
password-protected bank account, 225 (ex. 3.3)
pattern, 444–445
`pattern-match`, *474*
pattern matching, 454
  implementation, 473–475
  unification vs., 459, 461 *n*
pattern variable, 444
  representation of, 469, 484–486
`pc` register, 517
`perform` (in register machine), 499
  simulating, 527
`perform-action`, *527*
Perlis, Alan J., xv, 99 *n*
  quips, 7 *n*, 11 *n*
permutations of a set, 123
  `permutations`, *124*
Phillips, Hubert, 420 (ex. 4.42)

$\pi$ (pi)
  approximation with half-interval method, 68
  approximation with Monte Carlo integration, 228 (ex. 3.5), 354 (ex. 3.82)
  Cesàro estimate for, 226, 352
  Leibniz's series for, 57 *n*, 335
  stream of approximations, 335–337
  Wallis's formula for, 61 (ex. 1.31)
`pi-stream`, *335*
`pi-sum`, *57*
  with higher-order procedures, *59*
  with `lambda`, *62*
picture language, 126–141
Pingala, Áchárya, 46 *n*
pipelining, 298 *n*
Pitman, Kent M., 3 *n*
Planner, 416 *n*
point, represented as a pair, 89 (ex. 2.2)
pointer
  in box-and-pointer notation, 97
  typed, 535
`polar` package, 183
`polar?`, *176*
poly, 204
`polynomial` package, 204
polynomial(s), 202–216
  canonical form, 211
  dense, 208
  evaluating with Horner's rule, 119 (ex. 2.34)
  hierarchy of types, 210–211
  indeterminate of, 202
  sparse, 208
  univariate, 203
polynomial arithmetic, 202–216
  addition, 204–207
  division, 210 (ex. 2.91)
  Euclid's Algorithm, 212 *n*
  greatest common divisor, 212–214, 216 *n*
  interfaced to generic arithmetic system, 204
  multiplication, 204–207
  probabilistic algorithm for GCD, 216 *n*
  rational functions, 211–216
  subtraction, 209 (ex. 2.88)
`pop`, *517*
Portable Standard Lisp, 3 *n*
porting a language, 607
PowerPC, 315 *n*
power series, as stream, 332 (ex. 3.59)
  adding, 333 (ex. 3.60)
  dividing, 334 (ex. 3.62)
  integrating, 332 (ex. 3.59)
  multiplying, 333 (ex. 3.60)
predicate, 18
  of `cond` clause, 18
  of `if`, 19
  naming convention for, 24 *n*
prefix code, 162
prefix notation, 6
  infix notation vs., 151 (ex. 2.58)
`prepositions`, *422*
`preserving`, 572, 574 (ex. 5.31), *590*, 595 (ex. 5.37)
pretty-printing, 7
`prime?`, *50*, *330*
`prime-sum-pair`, *412*
`prime-sum-pairs`, *123*
  infinite stream, 338
prime number(s), 50–53
  cryptography and, 53 *n*
  Eratosthenes's sieve for, 327
  Fermat test for, 51–53
  infinite stream of (*see* `primes`)
  Miller-Rabin test for, 56 (ex. 1.28)
  testing for, 50–56
`primes` (infinite stream), *327*
  implicit definition, *330*
`primitive-apply`, *554*
`primitive-implementation`, *382*
`primitive-procedure?`, 377, *382*
`primitive-procedure-names`, *382*
`primitive-procedure-objects`, *382*
primitive constraints, 286
primitive expression, 4
  evaluation of, 10
  name of primitive procedure, 5
  name of variable, 7
  number, 5

primitive procedures (those marked *ns* are not in the IEEE Scheme standard)
`*`, *5*
`+`, *5*
`-`, *6*, *18 n*
`/`, *6*
`<`, *18*
`=`, *18*
`>`, *18*
`apply`, *183 n*
`atan`, *174 n*
`car`, *85*
`cdr`, *85*
`cons`, *85*
`cos`, *69*
`display`, *86 n*
`eq?`, *144*
`error` (*ns*), *68 n*
`eval` (*ns*), *387*
`list`, *100*
`log`, *70* (ex. 1.36)
`max`, *94*
`min`, *94*
`newline`, *86 n*
`not`, *19*
`null?`, *102*
`number?`, *147*
`pair?`, *109*
`quotient`, *332* (ex. 3.58)
`random` (*ns*), *52*, *229 n*
`read`, *383 n*
`remainder`, *45*
`round`, *201 n*
`runtime` (*ns*), *54* (ex. 1.22)
`set-car!`, *252*
`set-cdr!`, *252*
`sin`, *69*
`symbol?`, *148*
`vector-ref`, *534*
`vector-set!`, *534*
primitive query. *See* simple query
principle of least commitment, 175
`print` operation in register machine, 499
`print-point`, *90* (ex. 2.2)
`print-queue`, 266 (ex. 3.21)
`print-rat`, *86*
`print-result`, *561*
  monitored-stack version, *563*
`print-stack-statistics` operation in register machine, 530
printing, primitives for, 86 *n*
probabilistic algorithm, 52–53, 216 *n*, 327 *n*
`probe`
  in constraint system, *292*
  in digital-circuit simulator, *281*
`proc` register, 548
procedural abstraction, 26
procedural representation of data, 91–93
  mutable data, 260–261
procedure, 4
  anonymous, 62
  arbitrary number of arguments, 6, 104 (ex. 2.20)
  as argument, 57–61
  as black box, 26–27
  body of, 12
  compound, 12
  creating with `define`, 12
  creating with `lambda`, 62, 238, 240
  as data, 4
  definition of, 12–13
  first-class in Lisp, 77
  formal parameters of, 12
  as general method, 66–72
  generic, 166, 170
  higher-order (*see* higher-order procedure)
  implicit `begin` in body of, 221 *n*
  mathematical function vs., 21–22
  memoized, 272 (ex. 3.27)
  monitored, 224 (ex. 3.2)
  name of, 12
  naming (with `define`), 12
  as pattern for local evolution of a process, 31
  as returned value, 72–78
  returning multiple values, 521 *n*
  scope of formal parameters, 29
  special form vs., 401 (ex. 4.26), 409
`procedure-body`, *377*
`procedure-environment`, *377*
`procedure-parameters`, *377*

procedure application
  combination denoting, 6
  environment model of, 241–244
  substitution model of (*see* substitution model of procedure application)
process, 1
  iterative, 34
  linear iterative, 34
  linear recursive, 34
  local evolution of, 31
  order of growth of, 42
  recursive, 34
  resources required by, 42
  shape of, 34
  tree-recursive, 37–41
`product`, 60 (ex. 1.31)
  as accumulation, 61 (ex. 1.32)
`product?`, *148*
program, 1
  as abstract machine, 384
  comments in, 124*n*
  as data, 384–387
  incremental development of, 8
  structure of, 8, 26, 29–31 (*see also* abstraction barriers)
  structured with subroutines, 386*n*
program counter, 517
programming
  data-directed (*see* data-directed programming)
  demand-driven, 323
  elements of, 4–5
  functional (*see* functional programming)
  imperative, 234
  odious style, 325*n*
programming language, 1
  design of, 398
  functional, 356
  logic, 440
  object-oriented, 200*n*
  strongly typed, 351*n*
  very high-level, 22*n*
Prolog, 416*n*, 439*n*
`prompt-for-input`, *383*
prompts, 383
  explicit-control evaluator, 561
  lazy evaluator, 404
  metacircular evaluator, 383
  nondeterministic evaluator, 435
  query interpreter, 468
`propagate`, *281*
propagation of constraints, 285–296
proving programs correct, 22*n*
pseudodivision of polynomials, 214
pseudo-random sequence, 226*n*
pseudoremainder of polynomials, 214
`push`, *517*
`put`, 181, *271*
puzzles
  eight-queens puzzle, 124 (ex. 2.42), 420 (ex. 4.44)
  logic puzzles, 418–420
Pythagorean triples
  with nondeterministic programs, 417 (ex. 4.35), 417 (ex. 4.36), 418 (ex. 4.37)
  with streams, 342 (ex. 3.69)

`qeval`, 461, *470*
quantum mechanics, 357*n*
quasiquote, 575*n*
`queens`, *125* (ex. 2.42)
query, 441. *See also* simple query; compound query
`query-driver-loop`, *469*
query interpreter, 441
  adding rule or assertion, 462
  compound query (*see* compound query)
  data base, 479–482
  driver loop, 462, 468–470
  environment structure in, 490 (ex. 4.79)
  frame, 454, 486
  improvements to, 467 (ex. 4.67), 488 (ex. 4.76), 489 (ex. 4.77)
  infinite loops, 464–465, 467 (ex. 4.67)
  instantiation, 469–470
  Lisp interpreter vs., 460, 461, 489 (ex. 4.79)

overview, 453–462
pattern matching, 454, 473–475
pattern-variable representation, 469, 484–486
problems with `not` and `lisp-value`, 465–466, 489 (ex. 4.77)
query evaluator, 461, 470–473
rule (*see* rule)
simple query (*see* simple query)
stream operations, 482–483
streams of frames, 454, 462 *n*
syntax of query language, 483–486
unification, 458–459, 477–479
query language, 440, 441–453
abstraction in, 448
compound query (*see* compound query)
data base, 441–443
equality testing in, 448 *n*
extensions to, 467 (ex. 4.66), 488 (ex. 4.75)
logical deductions, 451–453
mathematical logic vs., 462–468
rule (*see* rule)
simple query (*see* simple query)
question mark, in predicate names, 24 *n*
queue, 261–266
double-ended, 266 (ex. 3.23)
front of, 261
operations on, 262
procedural implementation of, 266 (ex. 3.22)
rear of, 261
in simulation agenda, 283
quotation, 142–145
of character strings, 143 *n*
of Lisp data objects, 143
in natural language, 142
quotation mark, single vs. double, 143 *n*
`quote` (special form), *143 n*
`read` and, 383 *n*, 485 *n*
`quoted?`, *369*
`quotient` (primitive procedure), *332* (ex. 3.58)
Rabin, Michael O., 56 (ex. 1.28)
radicand, 23
Ramanujan, Srinivasa, 342 *n*
Ramanujan numbers, 342 (ex. 3.71)
`rand`, *226*
with reset, 229 (ex. 3.6)
`random` (primitive procedure), *52*
assignment needed for, 220 *n*
MIT Scheme, *229 n*
`random-in-range`, *229* (ex. 3.5)
`random-numbers` (infinite stream), *352*
random-number generator, 220 *n*, 225
in Monte Carlo simulation, 226
in primality testing, 51
with reset, 229 (ex. 3.6)
with reset, stream version, 353 (ex. 3.81)
Raphael, Bertram, 439 *n*
`rational` package, 190
rational function, 211–216
reducing to lowest terms, 214–216
rational number(s)
arithmetic operations on, 83–87
in MIT Scheme, 24 *n*
printing, 86
reducing to lowest terms, 87, 89
represented as pairs, 86
rational-number arithmetic, 83–87
interfaced to generic arithmetic system, 190
need for compound data, 80
Raymond, Eric, 399 *n*, 416 *n*
RC circuit, 344 (ex. 3.73)
`read` operation in register machine, 498
`read` (primitive procedure), *383 n*
dotted-tail notation handling by, 475
macro characters, 485 *n*
reader macro character, 485 *n*
read-eval-print loop, 7. *See also* driver loop
`read-eval-print-loop`, *561*

`real-part`
  data-directed, *184*
  polar representation, *174*
  rectangular representation, *174*
  with tagged data, *177*
`real-part-polar`, *177*
`real-part-rectangular`, *176*
real number, 5 *n*
`rear-ptr`, *263*
`receive` procedure, 521 *n*
record, in a data base, 160
rectangle, representing, 90 (ex. 2.3)
`rectangular` package, 182
`rectangular?`, *176*
recursion, 9
  data-directed, 207
  expressing complicated process, 9
  in rules, 449
  in working with trees, 108
recursion equations, 2
recursion theory, 386 *n*
recursive procedure
  recursive procedure definition, 26
  recursive process vs., 35
  specifying without `define`, 392 (ex. 4.21)
recursive process, 34
  iterative process vs., 32–36, 243 (ex. 3.9), 506, 594 (ex. 5.34)
  linear, 34, 43
  recursive procedure vs., 35
  register machine for, 506–512
  tree, 37–41, 43
red-black tree, 158 *n*
reducing to lowest terms, 87, 89, 214–216
Rees, Jonathan A., 374 *n*, 394 *n*
referential transparency, 233
`reg` (in register machine), 497
  simulating, 527
register(s), 491
  representing, 516
  tracing, 532 (ex. 5.18)
`register-exp`, *528*
`register-exp-reg`, *528*
register machine, 491
  actions, 498–499
  controller, 492–494
  controller diagram, 494
  data-path diagram, 493
  data paths, 492–494
  design of, 492–513
  language for describing, 494–499
  monitoring performance, 530–533
  simulator, 513–533
  stack, 506–512
  subroutine, 502–506
  test operation, 493
register-machine language
  `assign`, 497, 513
  `branch`, 496, 513
  `const`, 497, 512, 513
  entry point, 496
  `goto`, 496, 513
  instructions, 496, 512
  label, 496
  `label`, 496, 513
  `op`, 497, 513
  `perform`, 499, 513
  `reg`, 497, 512
  `restore`, 508, 513
  `save`, 508, 513
  `test`, 496, 513
register-machine simulator, 513–533
`registers-modified`, *588*
`registers-needed`, *588*
register table, in simulator, 517
relations, computing in terms of, 286, 438
relatively prime, 61 (ex. 1.33)
relativity, theory of, 316
release a mutex, 311
`remainder` (primitive procedure), *45*
`remainder-terms`, 213 (ex. 2.94)
remainder modulo *n*, 51
`remove`, *124*
`remove-first-agenda-item!`, 280, *285*
`require`, *414*
  as a special form, 437 (ex. 4.54)
reserved words, 598 (ex. 5.38), 603 (ex. 5.44)
resistance
  formula for parallel resistors, 93, 96
  tolerance of resistors, 93
resolution, Horn-clause, 439 *n*

resolution principle, 439 *n*
`rest-exps`, *371*
`rest-operands`, *372*
`rest-segments`, *283*
`rest-terms`, 205, *209*
`restore` (in register machine), 508, 529 (ex. 5.11)
  implementing, 538
  simulating, 527
`return` (linkage descriptor), 571
returning multiple values, 521 *n*
Reuter, Andreas, 314 *n*
`reverse`, 103 (ex. 2.18)
  as folding, 122 (ex. 2.39)
  rules, 468 (ex. 4.68)
Rhind Papyrus, 47 *n*
`right-branch`, *157*, *165*
`right-split`, *131*
ripple-carry adder, 278 (ex. 3.30)
Rivest, Ronald L., 53 *n*, 158 *n*
RLC circuit, 349 (ex. 3.80)
Robinson, J. A., 439 *n*
robustness, 141
rock songs, 1950s, 168 (ex. 2.70)
Rogers, William Barton, 127 *n*
`root` register, 541
roots of equation. *See* half-interval method; Newton's method
`rotate90`, *139*
`round` (primitive procedure), *201 n*
roundoff error, 5 *n*, 171 *n*
Rozas, Guillermo Juan, 586 *n*
RSA algorithm, 53 *n*
rule (query language), 448–453
  applying, 460–461, 475–477, 490 (ex. 4.79)
  without body, 449 *n*, 451, 473
Runkle, John Daniel, 128 *n*
`runtime` (primitive procedure), *54* (ex. 1.22)
Russian peasant method of multiplication, 47 *n*

`same` (rule), *448*
`same-variable?`, *148*, 204
sameness and change
  meaning of, 232–234
  shared data and, 257
satisfy a compound query, 446–448
satisfy a pattern (simple query), 445
`save` (in register machine), 508, 529 (ex. 5.11)
  implementing, 538
  simulating, 526
`scale-list`, *105*, *106*, *410*
`scale-stream`, *329*
`scale-tree`, *112*
`scale-vect`, 136 (ex. 2.46)
`scan` register, 542
`scan-out-defines`, 390 (ex. 4.16)
scanning out internal definitions, 389
  in compiler, 602 *n*, 603 (ex. 5.43)
Scheme, 3
  history of, 3 *n*
`scheme-number` package, 189
`scheme-number->complex`, *195*
`scheme-number->scheme-number`, *200* (ex. 2.81)
Scheme chip, 547, 548 (fig. 5.16)
Schmidt, Eric, 234 *n*
scope of a variable, 28. *See also* lexical scoping
  internal `define`, 388
  in `let`, 65
  procedure's formal parameters, 29
search
  of binary tree, 155
  depth-first, 416
  systematic, 415
`search`, *67*
secretary, importance of, 443
`segment-queue`, *283*
`segment-time`, *283*
`segments`, *283*
`segments->painter`, *137*
selector, 83
  as abstraction barrier, 88
  generic, 177, 179
`self-evaluating?`, *369*
self-evaluating expression, 364
semaphore, 311 *n*
  of size *n*, 313 (ex. 3.47)
semicolon, 11 *n*
  comment introduced by, 124 *n*
separator code, 162

sequence(s), 99
  as conventional interface, 113–126
  as source of modularity, 117
  operations on, 115–122
  represented by pairs, 99
`sequence->exp`, *372*
sequence accelerator, 336
sequence of expressions
  in consequent of `cond`, 19 *n*
  in procedure body, 12 *n*
`serialized-exchange`, *309*
  with deadlock avoidance, *314* (ex. 3.48)
serializer, 304–307
  implementing, 311–314
  with multiple shared resources, 307–311
series, summation of, 58
  accelerating sequence of approximations, 336
  with streams, 335
set, 151
  data base as, 160
  operations on, 151–152
  permutations of, 123
  represented as binary tree, 155–160
  represented as ordered list, 153–155
  represented as unordered list, 152–153
  subsets of, 113 (ex. 2.32)
`set!` (special form), *220*. *See also* assignment
  environment model of, 241 *n*
  value of, 220 *n*
`set-car!` (primitive procedure), *252*
  implemented with vectors, 537
  procedural implementation of, *261*
  value of, 252 *n*
`set-cdr!` (primitive procedure), *252*
  implemented with vectors, 537
  procedural implementation of, *261*
  value of, 252 *n*
`set-contents!`, *516*
`set-current-time!`, *283*
`set-front-ptr!`, *263*
`set-instruction-execution-proc!`, *522*
`set-rear-ptr!`, *263*
`set-register-contents!`, 514, *518*
`set-segments!`, *283*
`set-signal!`, 276, *280*
`set-value!`, 289, *294*
`set-variable-value!`, 378, *379*
`setup-environment`, *381*
shadow a binding, 237
Shamir, Adi, 53 *n*
shape of a process, 34
shared data, 257–260
shared resources, 307–311
shared state, 300
`shrink-to-upper-right`, *138*
Shrobe, Howard E., 440 *n*
side-effect bug, 234 *n*
sieve of Eratosthenes, 327
  `sieve`, *327*
$\sum$ (sigma) notation, 58
signal, digital, 273
`signal-error`, *561*
signal-flow diagram, 114, 344 (fig. 3.33)
signal processing
  smoothing a function, 78 (ex. 1.44)
  smoothing a signal, 345 (ex. 3.75), 346 (ex. 3.76)
  stream model of, 343–346
  zero crossings of a signal, 344 (ex. 3.74), 345 (ex. 3.75), 346 (ex. 3.76)
signal-processing view of computation, 114
`simple-query`, *470*
simple query, 443–446
  processing, 454, 455, 461, 470–471
simplification of algebraic expressions, 149
Simpson's Rule for numerical integration, 60 (ex. 1.29)
simulation
  of digital circuit (*see* digital-circuit simulation)
  event-driven, 273
  as machine-design tool, 563
  for monitoring performance of register machine, 530

Monte Carlo (*see* Monte Carlo simulation)
of register machine (*see* register-machine simulator)
`sin` (primitive procedure), *69*
sine
approximation for small angle, 44 (ex. 1.15)
power series for, 332 (ex. 3.59)
`singleton-stream`, *483*
SKETCHPAD, 286*n*
`smallest-divisor`, *50*
more efficient version, 54 (ex. 1.23)
Smalltalk, 286*n*
smoothing a function, 78 (ex. 1.44)
smoothing a signal, 345 (ex. 3.75), 346 (ex. 3.76)
snarf, 399*n*
Solar System's chaotic dynamics, 3*n*
Solomonoff, Ray, 226*n*
`solve` differential equation, *347*, *348*
lazy-list version, *411*
with scanned-out definitions, *391* (ex. 4.18)
solving equation. *See* half-interval method; Newton's method; `solve`
source language, 567
source program, 567
Spafford, Eugene H., 607*n*
sparse polynomial, 208
special form, 11
as derived expression in evaluator, 372
need for, 25 (ex. 1.6)
procedure vs., 401 (ex. 4.26), 409
special forms (those marked *ns* are not in the IEEE Scheme standard)
`and`, *19*
`begin`, *220*
`cond`, *17*
`cons-stream` (*ns*), *321*
`define`, *7*, *12*
`delay` (*ns*), *320*
`if`, *19*
`lambda`, *62*
`let`, *64*
`let*`, *375* (ex. 4.7)
`letrec`, *391* (ex. 4.20)
named `let`, *376* (ex. 4.8)
`or`, *19*
`quote`, *143n*
`set!`, *220*
`split`, 134 (ex. 2.45)
`sqrt`, *24*
block structured, *30*
in environment model, 249–250
as fixed point, *70*, *73*, *75*, *76*
as iterative improvement, 78 (ex. 1.46)
with Newton's method, *75*, *76*
register machine for, 502 (ex. 5.3)
as stream limit, *338* (ex. 3.64)
`sqrt-stream`, *335*
`square`, *12*
in environment model, 238–240
`square-limit`, *132*, *134*
`square-of-four`, *132*
`squarer` (constraint), 295 (ex. 3.34), 295 (ex. 3.35)
square root, 22–24. *See also* `sqrt`
stream of approximations, 334
`squash-inwards`, *139*
stack, 35*n*
framed, 550*n*
for recursion in register machine, 506–512
representing, 516, 538
stack allocation and tail recursion, 586*n*
`stack-inst-reg-name`, *527*
Stallman, Richard M., 286*n*, 416*n*
`start` register machine, 514, *518*
`start-eceval`, *605n*
`start-segment`, 89 (ex. 2.2), 137 (ex. 2.48)
state
local (*see* local state)
shared, 300
vanishes in stream formulation, 355
statements. *See* instruction sequence
`statements`, *588*
state variable, 34, 218
local, 219–225
Steele, Guy Lewis Jr., 3*n*, 36*n*, 235*n*, 286*n*, 399*n*, 416*n*

stop-and-copy garbage collector, 540–546
Stoy, Joseph E., 15 *n*, 47 *n*, 393 *n*
Strachey, Christopher, 76 *n*
stratified design, 140
stream(s), 218, 316–357
  delayed evaluation and, 346–350
  empty, 319
  implemented as delayed lists, 317–321
  implemented as lazy lists, 409–411
  implicit definition, 328–330
  infinite (*see* infinite streams)
  used in query interpreter, 454, 462 *n*
`stream-append`, *340*
`stream-append-delayed`, *482*
`stream-car`, 319, *321*
`stream-cdr`, 319, *321*
`stream-enumerate-interval`, *321*
`stream-filter`, *322*
`stream-flatmap`, *483*, 487 (ex. 4.74)
`stream-for-each`, *320*
`stream-limit`, 338 (ex. 3.64)
`stream-map`, *320*
  with multiple arguments, 325 (ex. 3.50)
`stream-null?`, 319
  in MIT Scheme, 319 *n*
`stream-ref`, *319*
`stream-withdraw`, *355*
strict, 400
string. *See* character string
strongly typed language, 351 *n*
`sub` (generic), *189*
`sub-complex`, *173*
`sub-interval`, 95 (ex. 2.8)
`sub-rat`, *84*
`sub-vect`, 136 (ex. 2.46)
subroutine in register machine, 502–506
`subsets` of a set, *113* (ex. 2.32)
substitution model of procedure application, 13–17, 236
  inadequacy of, 230–231
  shape of process, 33–35
subtype, 197
  multiple, 199
success continuation (nondeterministic evaluator), 427, 429
successive squaring, 45
`sum`, *58*
  as accumulation, 61 (ex. 1.32)
  iterative version, 60 (ex. 1.30)
`sum?`, *148*
`sum-cubes`, *57*
  with higher-order procedures, *59*
`sum-integers`, *57*
  with higher-order procedures, *59*
`sum-odd-squares`, *113*, *117*
`sum-of-squares`, *13*
  in environment model, 241–243
`sum-primes`, *318*
summation of a series, 58
  with streams, 335
supertype, 197
  multiple, 199
Sussman, Gerald Jay, 3 *n*, 36 *n*, 286 *n*, 416 *n*
Sussman, Julie Esther Mazel, nieces of, 142
Sutherland, Ivan, 286 *n*
symbol(s), 142
  equality of, 144
  interning, 537
  quotation of, 143
  representation of, 536
  uniqueness of, 257 *n*
`symbol?` (primitive procedure), *148*
  data types and, 193 (ex. 2.78)
  implemented with typed pointers, 538
`symbol-leaf`, *165*
symbolic algebra, 202–216
symbolic differentiation, 145–151, 184 (ex. 2.73)
symbolic expression, 82. *See also* symbol(s)
`symbols`, *165*
SYNC, 315 *n*
synchronization. *See* concurrency
syntactic analysis, separated from execution
  in metacircular evaluator, 393–398
  in register-machine simulator, 520, 525

syntactic sugar, 11 *n*
  `define`, 370
  `let` as, 65
  looping constructs as, 36
  procedure vs. data as, 279 *n*
syntax. *See also* special forms
  abstract (*see* abstract syntax)
  of expressions, describing, 12 *n*
  of a programming language, 11
syntax interface, 279 *n*
systematic search, 415

`#t`, *18 n*
table, 266–273
  backbone of, 267
  for coercion, 195
  for data-directed programming, 181
  local, 270–271
  *n*-dimensional, 272 (ex. 3.25)
  one-dimensional, 267–268
  operation-and-type (*see* operation-and-type table)
  represented as binary tree vs. unordered list, 272 (ex. 3.26)
  testing equality of keys, 272 (ex. 3.24)
  two-dimensional, 268–270
  used in simulation agenda, 283
  used to store computed values, 272 (ex. 3.27)
tableau, 337
tabulation, 41 *n*, 272 (ex. 3.27)
`tack-on-instruction-sequence`, *590*
`tagged-list?`, *369*
tagged architecture, 535 *n*
tagged data, 175–179, 535 *n*
tail recursion, 35
  compiler and, 586
  environment model of evaluation and, 244 *n*
  explicit-control evaluator and, 556–558, 564 (ex. 5.26), 565 (ex. 5.28)
  garbage collection and, 586 *n*
  metacircular evaluator and, 556
  in Scheme, 36 *n*
tail-recursive evaluator, 556
tangent
  as continued fraction, 72 (ex. 1.39)
  power series for, 334 (ex. 3.62)
target register, 571
Technological University of Eindhoven, 311 *n*
Teitelman, Warren, 3 *n*
terminal node of a tree, 9
term list of polynomial, 204
  representing, 207–209
`term-list`, *204*
`test` (in register machine), 496
  simulating, 525
`test-and-set!`, *312*, *313 n*
`test-condition`, *525*
test operation in register machine, 493
`text-of-quotation`, *369*
Thatcher, James W., 91 *n*
`the-cars`
  register, 537, 542
  vector, 535
`the-cdrs`
  register, 537, 542
  vector, 535
`the-empty-stream`, 319
  in MIT Scheme, 319 *n*
`the-empty-termlist`, 205, *209*
`the-global-environment`, *381*, *561 n*
THE Multiprogramming System, 311 *n*
theorem proving (automatic), 438 *n*
$\theta(f(n))$ (theta of $f(n)$), 43
thunk, 401–402
  call-by-name, 324 *n*
  call-by-need, 324 *n*
  forcing, 401
  implementation of, 404–406
  origin of name, 401 *n*
time
  assignment and, 297
  communication and, 316
  in concurrent systems, 298–303
  functional programming and, 354–357
  in nondeterministic computing, 413, 415
  purpose of, 298 *n*

`timed-prime-test`, *54* (ex. 1.22)
time segment, in agenda, 283
time slicing, 313
timing diagram, 301 (fig. 3.29)
TK!Solver, 286 *n*
tower of types, 197 (fig. 2.25)
tracing
  instruction execution, 532 (ex. 5.16)
  register assignment, 532 (ex. 5.18)
`transform-painter`, *138*
transparency, referential, 233
`transpose` a matrix, 121 (ex. 2.37)
tree
  binary, 155 (*see also* binary tree)
  B-tree, 158 *n*
  combination viewed as, 9
  counting leaves of, 108
  enumerating leaves of, 116
  fringe of, 111 (ex. 2.28)
  Huffman, 162
  lazy, 410 *n*
  mapping over, 112–113
  red-black, 158 *n*
  represented as pairs, 107–111
  reversing at all levels, 110 (ex. 2.27)
`tree->list...`, *158* (ex. 2.63)
`tree-map`, 113 (ex. 2.31)
tree accumulation, 10
tree-recursive process, 37–41
  order of growth, 43
trigonometric relations, 174
true, 18 *n*
`true`, *18 n*
`true?`, *377*
truncation error, 5 *n*
truth maintenance, 416 *n*
`try-again`, *416*
Turing, Alan M., 386 *n*, 387 *n*
Turing machine, 386 *n*
Turner, David, 122 *n*, 340 *n*, 356 *n*
type(s)
  cross-type operations, 194
  dispatching on, 179
  hierarchy in symbolic algebra, 210–211
  hierarchy of, 197–202
  lowering, 198, 201 (ex. 2.85)
  multiple subtype and supertype, 199
  raising, 198, 201 (ex. 2.83)
  subtype, 197
  supertype, 197
  tower of, 197 (fig. 2.25)
typed pointer, 535
type field, 535 *n*
type-inferencing mechanism, 351 *n*
type tag, 170, 175
  two-level, 192
`type-tag`, *176*
  using Scheme data types, 193 (ex. 2.78)
typing input expressions, 7 *n*

unbound variable, 237
`unev` register, 548
unification, 458–459
  discovery of algorithm, 439 *n*
  implementation, 477–479
  pattern matching vs., 459, 461 *n*
`unify-match`, *477*
`union-set`, 152
  binary-tree representation, 160 (ex. 2.65)
  ordered-list representation, 155 (ex. 2.62)
  unordered-list representation, 153 (ex. 2.59)
`unique` (query language), 488 (ex. 4.75)
`unique-pairs`, 124 (ex. 2.40)
unit square, 134
univariate polynomial, 203
universal machine, 386
  explicit-control evaluator as, 566
  general-purpose computer as, 566
University of California at Berkeley, 3 *n*
University of Edinburgh, 439 *n*
University of Marseille, 439 *n*
UNIX, 565 *n*, 607 *n*
`unknown-expression-type`, *561*
`unknown-procedure-type`, *561*
unordered-list representation of sets, 152–153

unspecified values
  `define`, 8 *n*
  `display`, 86 *n*
  `if` without alternative, 284 *n*
  `newline`, 86 *n*
  `set!`, 220 *n*
  `set-car!`, 252 *n*
  `set-cdr!`, 252 *n*
`up-split`, 132 (ex. 2.44)
`update-insts!`, *522*
`upper-bound`, 94 (ex. 2.7)
upward compatibility, 408 (ex. 4.31)
`user-initial-environment` (MIT Scheme), *387 n*
`user-print`, *383*
  modified for compiled code, *605 n*

V operation on semaphore, 311 *n*
`val` register, 548
value
  of a combination, 6
  of an expression, 7 *n* (*see also* unspecified values)
`value-proc`, 524
variable, 7. *See also* local variable
  bound, 28
  free, 28
  scope of, 28 (*see also* scope of a variable)
  unbound, 237
  value of, 7, 237
`variable`, *204*
`variable?`, *148*, *369*
variable-length code, 161
vector (data structure), 534
vector (mathematical)
  operations on, 120 (ex. 2.37), 136 (ex. 2.46)
  in picture-language frame, 134
  represented as pair, 136 (ex. 2.46)
  represented as sequence, 120 (ex. 2.37)
`vector-ref` (primitive procedure), *534*
`vector-set!` (primitive procedure), *534*
Venus, 142 *n*
`verbs`, *421*
very high-level language, 22 *n*

Wadler, Philip, 234 *n*
Wadsworth, Christopher, 351 *n*
Wagner, Eric G., 91 *n*
Walker, Francis Amasa, 128 *n*
Wallis, John, 61 *n*
Wand, Mitchell, 361 *n*, 552 *n*
Waters, Richard C., 118 *n*
`weight`, *165*
`weight-leaf`, *165*
Weyl, Hermann, 79
"what is" vs. "how to" description. *See* declarative vs. imperative knowledge
`wheel` (rule), *449*, 467 (ex. 4.65)
`width`, *95*
width of an interval, 95 (ex. 2.9)
Wilde, Oscar (Perlis's paraphrase of), 7 *n*
Wiles, Andrew, 51 *n*
Winograd, Terry, 416 *n*
Winston, Patrick Henry, 416 *n*, 426 *n*
wire, in digital circuit, 273
Wisdom, Jack, 3 *n*
Wise, David S., 324 *n*
wishful thinking, 84, 146
`withdraw`, *220*
  problems in concurrent system, 299
`without-interrupts`, 313 *n*
world line of a particle, 317 *n*, 355 *n*
Wright, E. M., 330 *n*
Wright, Jesse B., 91 *n*

`xcor-vect`, 136 (ex. 2.46)
Xerox Palo Alto Research Center, 3 *n*, 286 *n*

*Y* operator, 393 *n*
`ycor-vect`, 136 (ex. 2.46)
Yochelson, Jerome C., 541 *n*

Zabih, Ramin, 416 *n*
zero crossings of a signal, 344 (ex. 3.74), 345 (ex. 3.75), 346 (ex. 3.76)
zero test (generic), 193 (ex. 2.80)
  for polynomials, 209 (ex. 2.87)
Zetalisp, 3 *n*
Zilles, Stephen N., 91 *n*
Zippel, Richard E., 216 *n*