

Bibliography

- American National Standards Institute (1983). *Military Standard Ada Programming Language*. ANSI/MIL-STD-1815A-1983.
- Backus, J. (1978a). "The History of FORTRAN I, II, and III." *SIGPLAN Notices* 13, 8 (August 1978), pp. 165–180.
- Backus, J. (1978b). "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs." *Commun. ACM* 21, 8 (August 1978), pp. 613–641.
- Berry, D. M. (1971). "Introduction to Oregano." *SIGPLAN Notices* 6, 2 (February 1971), pp. 171–190.
- Clocksin, W.F., and Mellish, C.S. (1984). *Programming in Prolog*, second edition, Springer-Verlag, 1984.
- Cohen, J. (1981). "Garbage Collection of Linked Data Structures." *ACM Computing Surveys* 13, 3 (September 1981), pp. 341–367.
- DeRemer, F., and Kron, H. (1976). "Programming-in-the-Large Versus Programming-in-the-Small." *IEEE Transact. Software Eng. SE-2* (June 1975), pp. 80–86.
- Dijkstra, E.W. (1968). "Go To Statement Considered Harmful." *Commun. ACM* 11, 3 (March 1968), pp. 147–148.
- Goldberg, A. (1984). *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley, 1984.
- Goldberg, A., and Robson, D. (1983). *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.
- Intermetrics, Inc. (1995). *Ada 95 Rationale: The Language, The Standard Libraries*. Intermetrics, 1995.
- International Standards Organization (1982). *Specification for Computer Programming Language Pascal*. ISO 7185-1982, 1982.
- International Standards Organization (1995). *Ada Reference Manual: Language and Standard Libraries*. ISO/IEC 8652:1995(E).
- Jensen, K., Wirth, N., Mickel, A.B., and Miner, J.F. (1985). *Pascal User Manual and Report*, third edition. Springer-Verlag, 1985.
- Johnston, J.B. (1971). "The Contour Model of Block Structured Processes." *SIGPLAN Notices* 6, 2 (June 1971), pp. 55–82.
- Kowalski, R. (1979). "Algorithm = Logic + Control." *Commun. ACM* 22, 7 (July 1979), pp. 424–436.
- MacLennan, B.J. (1990). *Functional Programming: Practice and Theory*. Addison-Wesley, 1990.
- MacLennan, B.J. (1983). "Values and Objects in Programming Languages." *SIGPLAN Notices* 17, 12 (December 1983), pp. 70–79; reprinted in Gerald E. Peterson (ed.), *Object-Oriented Computing, Volume 1: Concepts*, pp. 9–14. IEEE Computer Society Press, 1987.
- Manna, Z., and Waldinger, R. (1985). *The Logical Basis for Computer Programming*. Addison-Wesley, 1985.

- McCarthy, J. (1960). "Recursive Functions of Symbolic Expressions and Their Computation by Machine." *Commun. ACM* 3, 4 (April 1960), pp. 184-195.
- McCarthy, J. (1978). "History of LISP." *SIGPLAN Notices* 13, 8 (August 1978), pp. 217-223.
- Naur, P. (1978). "The European Side of the Last Phase of the Development of Algol 60." *SIGPLAN Notices* 13, 8 (August 1978), pp. 15-44.
- Perlis, A.J. (1978). "The American Side of the Development of Algol." *SIGPLAN Notices* 13, 8 (August 1978), pp. 3-14.
- Steele, G. L., Jr. (1984). *Common LISP: The Language*. Digital Press, 1984.
- United States Department of Defense (1980). *Reference Manual for the Ada Programming Language*, July 1980.
- Wirth, N. (1971). "The Programming Language Pascal." *Acta Inform.* 1 (1971), pp. 35-63.
- Wirth, N. (1975). "An Assessment of the Programming Language Pascal." *SIGPLAN Notices* 10, 6 (June 1975), pp. 23-30.
- Wulf, W., and Shaw, M. (1973). "Global Variable Considered Harmful." *SIGPLAN Notices* 8, 2 (February 1973), pp. 28-34.

Index

- (period)
 - Pascal and Ada (selection), 184, 266, 269, 272, 274
 - Prolog (cons), 453–55
- .. (double period, “up to”), 175, 252
- ; (semicolon)
 - Prolog, 448, 468, 484
 - terminating vs. separating, 300–1
- : (colon)
 - functional programming (application), 360
 - Smalltalk, 425, 427
- ’ (LISP quote, “quote”), 315, 371. *See also* quote (LISP)
- ‘ (single quotes), 21n
- (...) (parentheses), 314–15, 319–20
- {...} (braces in BNF, “either”), 153
- [...] (square brackets)
 - Algol, 97, 118
 - BNF (“optional”), 154
 - functional programming (construction), 362
 - Pascal, 177, 180–81
 - Prolog (list), 453–55
 - Smalltalk, 408, 427
- <...> (angle brackets), 150, 360
- <> (less-greater)
 - set inequality, 177
 - unconstrained, 254–55
- <, > (less and greater relations), 174–75, 177
- = (equality or assignment), 99–100, 207, 471, 485–86
- == (equality), 100
- := (assignment, “gets” or “becomes”), 99–100
 - default parameter, 290
- ::= (“is defined as”), 150
- .. (Prolog), 481
- :- (“if”), 447, 450, 450n
- ← (left-arrow, “gets” or “if”), 405, 450n
- ↑ (up-arrow), 99, 145, 189–91, 454–55
- ↑ (“return”), 411
- >, >=, 65, 289
- <=, >= (subset, superset), 177, 179
- \= (“not equal”), 447, 456, 482, 485–86, 488–89
- × (code duplication), 218
- / (reduction, “reduce by”), 360–61
- | (“cut”), 476–80
- | (vertical line), 151, 454n
- .. (local variables), 411
- (composition, “composed with”), 360–61
- @ (at-sign, display point), 406, 417
- + (plus, “union”), 177
- + (superscript plus, Kleene cross), 153
- (minus), 177
- (Ada comment), 265
- * (asterisk)
 - multiplication, 50
 - set intersection, 177
- * (superscript star, Kleene star), 153, 156
- ** (FORTRAN exponentiation), 50
- (bar over constant, “constant”), 360
- ₁₀ (subten, “times 10 to the”), 148
- α (alpha)
 - array addressing (“address of”), 71
 - functional programming (“apply to all”), 360
- 1, 2, ...** (boldface numbers, “first,” “second,” ...), 360, 362
- abstract (Ada reserved word), 439
- Abstract vs. concrete operations, 244
- Abstract data type, 67–68, 174, 244, 253–54, 264–66, 319–20, 323, 327–28, 395, 412, 416–17, 443n, 452, 459
- Abstract declaration or subprogram (Ada), 439–41
- Abstraction, procedural or control. *See* Procedures
- Abstraction Principle, 17–18, 54–56, 105, 182, 194, 200, 271, 351, 356, 365, 400, 409, 412–13, 418–20, 423, 442, 496
- Abstract type language, 244, 305
- accept-statement, 294–99
- Access path, 185
- Access rights, 262–63
- Access types, 250, 269, 440. *See also* Pointer types
- Accuracy constraint, 254. *See also* Numeric data types
- Action and focus, 34, 77, 395
- Activation and deactivation, procedure, 62, 128, 220, 235, 238, 240, 294, 434–35
- Activation records, 61–66, 113–14, 127–28, 162, 211–40, 432–36
 - block, 114, 235–38
 - defined, 62
 - nonrecursive procedure, 63
 - recursive procedure, 212–14, 221, 232, 234
 - Smalltalk, 432–36
 - summarized, 239
- Active vs. passive, 421–22
- Actual parameter, 55, 162, 222, 227
- Ada, 117, 139, 169, 182–83, 188, 192, 194, 203, 205, 243–306, 366, 395, 399, 422–25, 427, 443
 - Ada 95 language, 246, 286, 297, 303, 424, 436–41
 - complexity criticized, 303–4
 - history, 243–46, 303–4
 - goals, 245
 - name, 245
 - Rationale, 306
 - trademarked, 246
- addprop (LISP function), 330, 332
- Addresses. *See* Pointer types; References and referencing
- Addressing equation, 72
- Address modification, 9–10
- add1 (LISP function), 317–18
- add1-map (LISP function), 346
- adjoin (predicate), 486–87
- Aesthetics. *See* Elegance; Elegance Principle
- Agents, objects as, 426, 442
- AI, 309–12, 394–95, 446, 465
- Algebraic notation, 43, 88–89, 359, 362
- Algol-like language, 147, 162–63, 171
- Algol-W, 170, 183, 208, 446
- Algol-58, 95–96, 119, 162, 301
- Algol-60, 95–164, 179–81, 189, 194–97, 201–2, 205, 211, 228, 235–236, 249, 286, 299–301, 303–6, 344, 364, 366, 404, 436
 - hardware for, 163
 - history, 95–97, 99–100, 123–24, 144–45, 161–63, 167, 170, 173
 - improvement on successors, 162–63
 - racehorse rather than camel, 97
- Algol-68, 69, 117, 119, 162, 165, 170, 206–7, 251
- Aliasing, 82–85, 188, 202–3, 338–40
- a-list, 326–27, 378
- Allocation, static vs. dynamic, 76. *See also* Storage management
- all-pairs (LISP function), 347, 354–55, 357
- Alpha (Greek letter)
 - array addressing (“address of”), 71
 - functional programming (“apply to all”), 360
- Alphanumeric, 153
- Alphard, 244
- Alternates, 121, 379
 - in BNF, 151, 153–54
- Ampliative vs. reductive, 33–34, 162, 316, 396, 421, 480
- Analogies, 156–57, 493
- ancestor (predicate), 447
- and (LISP function), 344–45
- Angle brackets, 150, 360
- Annexes, special needs (Ada), 246, 303
- Anonymous
 - function, 353–55
 - procedure, block as anonymous, 238
 - type, 192, 252
- ANSI (American National Standards Institute). *See* Standards
- AP (activation record pointer), 218
- APL, 250, 359, 372
- append
 - LISP function, 328–30, 384, 454
 - Prolog predicate, 454–55, 484
- Apple computers, 405
- Application, function, 313, 339, 376–77
- Applicative language, 313, 339, 400. *See also* Functional programming; Function-oriented programming
- Applied value property, 332–33
- apply (LISP function), 380–85, 386–88
- apval (LISP property), 332–33, 343, 363
- AR (activation record), 64
- Arithmetic expression tree, 335, 455–58
- Arithmetic IF-statement, 46
- Arithmetic operations, 11–13, 68–69, 250, 253, 318, 417, 425–26, 471–74
- Arrays, 17–18, 70–75, 118–19, 179–83, 254–55. *See also* Indexing

- Ada, 254–55
 Algol, 112–14, 118–19
 base type, 180–81
 bounds checking, 29, 58, 180
 conformant array schema, 183
 declaration, 42, 98–99
 dimensions, 71, 74–75, 118, 146, 181–83
 dynamic, 99, 118–19, 162, 223n, 234
 flexible, 119
 FORTRAN, 70–75
 implementation, 71–75
 index constraint, 254–55
 index type, 180–81
 initialization, 42
 lower bounds, 98, 118
 parameters, 57–58, 128–29, 182–83, 254–55
 Pascal, 179–83
 passed by reference, 57–58
 passed by value, 128–29
 pseudo-code, 17–18
 row- vs. column-major order, 72
- Arrow symbols. *See* Symbols at beginning of index
- Artificial intelligence, 309–12, 394–95, 446, 465
- Assembler, 9
 structured, 206
- asserta (predicate), 475, 477–79
- Assertions (Prolog), 447
- assertz (predicate), 475
- Assigned GOTO-statement (FORTRAN), 49–51
- Assignment statement, 16, 43, 400, 405
 record assignment, 184–85, 191–92
 symbol, 99–100, 207, 290
- ASSIGN-statement (FORTRAN), 49–50
- Assistant, Programmer's, 397–99
- assoc (LISP function), 327, 378, 384
- Association lists, 326–27, 378
- Asterisk
 vs. double asterisk, 50
 on exercises, 16n
 Kleene star, 153, 156
 set intersection, 177
- atom (LISP function), 319–20
- atomic (Prolog predicate), 456–57
- Atoms, 314–15, 317–19, 331–34, 343, 486
 etymology, 317
 nonnumeric, 318–19, 378–79
 numeric, 317–18, 377–78
 properties, 331–34, 340–41, 363–64, 442
 representation, 340–41
 simulate objects, 331–34, 411–12
- At-sign, 406, 417
- Automatic coding, 37, 39
- Automatic deduction or theorem proving, 446, 459–60, 466, 469, 475–76
- Automation Principle, 10, 17, 456, 496
- B (programming language), 206–7
- Babbage, Charles, 245
- Backtracking, 466–69, 475–80, 484–85
- Backus, John, 39–40, 96, 148–50, 359–63
- Backus-Naur (or Normal) Form. *See* BNF
- Backus's FP language, 359–63
- Backwards execution, 452, 466–69, 472–74
- Baroque language or feature, 138, 140, 164, 206
- Bar symbols. *See* Symbols at beginning of index
- Base language, 168
- Base type, 176, 180–81, 191
- BASIC, 118, 250
- BCPL, 206–7
- begin-end.** *See* Block; Compound statement
- Bell Labs, 11, 206–7
- Billington, David P., 157–60
- Binary
 operator, 356
 representation of numbers, 67
 reversing arguments, 357
 to unary, 355–56, 360
- BINARY FIXED (31), 249
- Binding. *See also* Declarations
 argument, 355–57
 class. *See* Class
 formal parameter, 55
 object, 405
 Prolog instantiation, 454, 472
 temporary, 364–67
- Binding constructs (bindings). *See* Declarations
- Binding time, 29, 42–44, 100–101, 138, 197, 441
 parameter inspection time, 134
- Blanks, ignoring, 86–87
- Blocks and block structure
 Ada, 300
 Algol, 101–7, 112–14, 162
 vs. compound statement, 101–2, 124
 implementation, 235–39
 limitations, 258–63
 LISP, 364–67
 Pascal, no blocks in, 196
 Smalltalk, 427
 stack storage management, 112–14
 two-coordinate addressing, 215–17, 240, 433
- BNF, 96–97, 124, 148–56, 163
 Chomsky type 2 grammar, 155
 context-free grammar, 154–56
 etymology, 150
 extended, 153–54
 grammar size, 303–5
- Body, 248, 265
 of Prolog clause, 450, 465
- Boldface, 97, 145, 171, 362
- Boole, George, 66n
- Boolean type and values, 66, 98, 115, 319
- Bottom-up control, 463–65
- Bounds checking, 29, 58, 180
- Box (class or package), 409–11, 416–17, 423–24, 431, 437–40
- Braces, curly, 153
- Bracketing, statement, 123–24, 301–2, 408
- Brackets. *See also* Symbols at beginning of index
 angle (()), 150, 360
 curly ({}), 153
 round, 314–15, 319–20, 370–72
 square ([]), 97, 118, 154, 176, 180–81, 362, 408, 427, 453–55
- Breadth-first search, 474–75
- Breakpoints, 26
- Bridge, Tacoma Narrows, 158–59
- Bruner, Jerome, 404
- bu (functional), 355–58, 360, 370
- Buffer, 267–68, 296–97
- Burroughs B5500, 163
- bu2, bu3, etc. (LISP functionals), 357
- By-copy and by-reference parameters, 286n.
See also Parameter passing modes
- Byron, Lord George Gordon (poet), 245
- C (programming language), 100, 117, 206–8, 251, 304, 389–90, 399, 436, 441
- cadr, caddr, cadar, etc. (LISP functions), 322, 360
- Calculus (Prolog example), 451, 455–58
- Call, implementation. *See* Procedures, implementation
- call (Prolog)
 predicate, 481
 trace, 470
- Callee vs. caller, 55
- Call in environment of caller or of definition.
See Scoping, static vs. dynamic
- Cambridge Polish notation, 313, 318
- Capitals, 31n, 450
- car (LISP function), 320–23, 360
 etymology, 337
 Prolog definition, 452–55
- Card columns, 30–31, 86–87, 143–44
- Cartesian product, 347–48
- Case selection, 47, 139–40, 199–201
 default, 210
 fall-through, 140
 implementations, 201
 switch-declaration, 98–99, 139–40, 199
- Casting. *See* Type conversions
- Catchers (LISP), 388n
- Categories, syntactic, 150. *See also* Nonterminal symbols
- cdr (LISP function), 321–23, 360
 etymology, 337
 Prolog definition, 452–55
- Cdr-encoding, 342
- char, Pascal data type, 173
- Character data type, 70, 173, 255–56. *See also* Atoms; Strings
- Characteristic (floating-point numbers), 67
- Character set, 144–45, 255. *See also* Lexics
- Character strings, 69–70
- Chomsky, Noam, 154–55
- Chomsky hierarchy, 154–55
- Circle (small), composition, 360–61
- Circular structures, 340, 391–92, 429
- Class, 163, 244, 277, 404, 408–21, 431–32, 441, 444. *See also* Packages; Subclass
- class (Ada type), 440
- class (Smalltalk class), 431
 Smalltalk built-in, 415
- Class method, 410–11
- Class protocol, 412, 416–17
- Class variable, 431
- Clauses (Prolog), 447, 449–50, 469–71
 Horn, 450, 482–83, 489
 order of, 450, 461
- CLOS (Common LISP Object System), 421, 441–42
- Closed-world assumption, 458–59, 482
- Closures, 225–28, 240, 385–88
- CLU, 244
- COBOL, 66, 86, 91, 117, 161, 167, 183, 250
- Code vs. data, 9–10, 315–16, 421–22
- Code generation or synthesis (compilation), 44, 248
- Coding conventions, 16, 125
- Coercions, 68–69, 116, 120, 417–18, 441
- Cognitive models, 14, 134–35, 321
- Cognitive science, 134–35, 404
- Colmerauer, Alain, 446, 460
- Colon symbols. *See* Symbols at beginning of index
- Column-major order, 72
- Columns, card, 30–31, 86–87, 143–44
- Combinator, 357–58. *See also* Functional
- Combinatorial power, 357–58
- Combining form, 357–58. *See also* Functional
- COMMON block, 81–84, 268
- Common LISP, 312, 313n, 315, 318, 399–400, 421, 440. *See also* LISP
- comp (LISP functional), 358, 360
- Comparisons, 15

- Compatibility, upward, 305
 Compatible types, 253
 Compilation, 56, 195, 248, 265–66, 274–76, 279, 291
 Compilers, 11, 30–33, 36–37, 43–44, 86, 248
 compatible, 395, 398
 implemented in own language, 206
 validation, 246
 Compile-time vs. run-time, 11, 36–37, 41–44, 100, 215–18
 Compiling routines. *See* Compilers
 Complex data type, 66, 115–16, 264–65, 416
 Complexity, 183, 204, 250, 256, 440–41. *See also* Simplicity Principle
 and elegance, 158
 featuritis, 304–5, 441
 metric, 303–5
 of programming, 7–8
 Components, record, 183–84
 Composite types, 286
 Composition, function, 358, 360–61
 Compound statement, 101–2, 124–25, 301
 Compound term. *See* Term (Prolog)
 Computed GOTO, 47, 199, 201
 Computer language, vs. programming language, 1
 Conceptual models, 14, 134–35, 321
 Concrete vs. abstract operations, 244
 Concurrency, 240, 245, 291–99, 305–6, 427–28, 435–36, 442, 465–66
 Concurrent Pascal, 205, 306
 cond (LISP conditional), 314, 344
 Condition, 198, 379
 Prolog clause, 450
 Conditional logical connectives, 344–45, 349–50, 379
 Conditional selection, 15, 46–48, 127, 310, 343–45
 Algol vs. FORTRAN, 121–23
 conditional expression, 127, 140, 310, 343–44
 consequent and alternate, 121, 379
 dangling **else** problem, 147
 LISP. *See* cond (LISP function); *if* (LISP function)
 multiway, 302
 Conformant array schema, 183
 cons (LISP function), 320, 327–29, 337, 388–89
 Prolog definition, 452–55
 Consequent, 121, 379
 Consistency, automatically maintained, 398
 const
 LISP functional, 358–60
 Pascal constant declaration, 193
 constant (Ada), 256
 Constant, changed via reference parameter, 59–60
 Constant declarations, 172, 193–94, 256–57
 Constant function, 358–60
 Constraint
 accuracy, 254
 discriminant, 254
 fixed-point, 250
 index, 254–55
 range. *See* Range constraint
 Construct (declarative vs. imperative), 41–42, 98–99
 Constructor (functional), 362
 Constructor, inverse, for selection, 454–55
 Constructors vs. primitives, 70, 75, 101, 320, 327–28
 consval (LISP function), 353, 355–56, 383
 Context, 78, 103–4, 111–12, 212, 222, 237–38, 240, 375, 378, 382–88. *See also* Environment; Name structure; Scoping
 Contextual error correction, 396
 Context clause, 266
 Context-free grammar and language, 154–56
 Context-sensitive grammar and language, 154
 Contextual error correction, 396
 CONTINUE statement, 51–52, 123
 Continuous data types. *See* Floating-point numbers.
 Contour diagram, 79–80, 99, 102–103, 214.
 See also Name structure; Scoping;
 Visibility
 dynamic vs. static scoping, 107–9, 368–69
 Control abstractions. *See* Procedures
 Controlled variable, 51
 Control vs. logic, 461, 469, 473–74, 480, 489–91. *See also* Bottom-up control; Top-down control
 Control structure, 44–65, 92, 121–40, 164, 196–204, 281–85, 408, 461, 489. *See also specific control structures*
 Conventions, 31n. *See also* Coding conventions
 Conversions, type, 68–69, 120, 250, 255. *See also* Coercions
 Copy-restore, 60–61, 286–88
 Copy rule, 56–57, 129–35, 140
 Core language (Ada), 246, 303
 Coroutines, 240
 Coupling, loose vs. tight, 296
 cousin (predicate), 447
 CPL, 206–7, 210
 Creeping Feature Syndrome, 304–5, 441
 Cross, Kleene, 153
 Culture, 36. *See also* Conventions; Style
 Cut (Prolog), 476–80, 483
 Cyclic structures, 340, 391–92, 429
 C++ (programming language), 207, 304, 421, 441

 D (display element), 231
 d (predicate), 451
 Dangling **else** problem, 147
 Dangling pointer or reference, 292–93, 389
 Dash, double (Ada comment), 265. *See also* Minus sign
 Data abstraction language, 244, 305
 Data vs. program, 9–10, 315–16, 421–22
 DATA statement, 42
 Data structure, 66–75, 92, 115–21, 163, 172–93, 248–56, 317–41, 450–61. *See also specific data structures*
 manager, 268–69
 Data traps, 26
 Data type, abstract. *See* Abstract data type
 date (record type), 184
 DayOfMonth (data type), 175–76
 DayOfWeek (data type), 174
 Deactivation. *See* Activation and deactivation, procedure
 Dead-lock, 295
 Debugging, 24–26
 Decimal numbers, 33, 39
 Decimal point, 145
 declare (Ada), 266, 269, 274, 277–78, 283, 300, 366
 Declarations, 76–77, 193, 257. *See also* Executable statements vs. declarations
 Ada, 247, 302
 binding constructs (bindings), 75–76, 193
 class, 409–421, 440
 constant, 172, 193–94, 256–57
 deferred, 257, 265
 definition vs. specification, 248
 dynamic in LISP, 317
 FORTRAN, 42
 implicit bindings (enumeration types), 173–75, 255–56
 implicit bindings (**for-loop**), 283
 LISP, 316–17, 363–67
 module, 264–78, 409–415, 436–440
 object (Ada), 248
 optional, 77, 87
 order, Pascal, 195–96
 overloading. *See* Overloading package, 264–78, 436–440
 Pascal, 193–96
 procedure, 54–56, 78–80, 99, 127, 172, 194–96, 257, 340, 409–11
 switch, 98–99, 139–40
 task, 248
 type, 172, 191–3, 196, 248–54
 variable, 28–32, 75–78, 127–28, 172, 256, 363–64, 405, 412
 Declarative vs. imperative. *See also* Executable statements vs. declarations;
 Imperative
 languages, 313
 statement, 98–99
 Decoding, instruction, 23–24
 Deduction, automatic, 446, 459–60, 466, 469, 475–76
 Defense in Depth Principle, 50–51, 87, 496
 Deferred constant, 257, 265
 define (LISP function), 370–71
 Definitions. *See* Declarations
 defun (LISP pseudo-function), 314, 317, 363–64, 370–71, 391n
 delprop (LISP function), 330
 delta, 250
 Denotations
 integer, 71
 numeric, 148–54
 Department of Defense, U.S., 244–46, 303–4
 Dependencies, implicit, 193–94
 Depth-first search, 469–71, 474–75, 487
 DeRemer, F., 56n
 Derived type, 251–54, 438
 Descriptive tools, 148–50
 Design, feature vs. language, 305
 Design, responsible, 114–15, 188–89, 389, 496
 Designer's model, 135
 Dewey, John, 404
 Diagnostics in extensible languages, 169
 Diagrams, 148
 contour. *See* Contour diagram
 Dialects. *See* Subset, language; Superset, language
 dif (predicate), 451–52, 455
 difference (LISP function), 317
 Difference, set, 177
 Differentiation, symbolic, 451, 455–58
 digits, 249
 Dijkstra, E. W., 48–49, 90, 126, 167–68, 243
 DIMENSION statement, 42
 Discrete type, 176, 180. *See also* char; Character data type; Integers
 Discriminant
 of quadratic equation, 110
 of tagged type, 437
 variant record, 187, 251
 Discriminant constraint, 254
 Discriminated union, 251. *See also* Variant record
 Display method, 162, 231–34, 238–39
 Display object (class), 413–15, 418–20, 436–40
 dispose, 263, 389
 distl (function), 347, 354–56, 361

- distr (function), 362
 Distribute left, 347, 354–56, 361
 Distribute right, 362
 DL (part of activation record). *See* Dynamic link
 DoD, 244–46, 303–4
 DO-loop, 51–53, 73–75, 85. *See also* Iteration, definite
 vs. Algol for-loop, 122
 nesting, 123
 syntactic problems, 86–87
 Domination (numeric types), 69
 Dot notation. *See* Period, single
 Dotted pair, 326n, 453–55
 Double precision numbers, 66, 115. *See also* Numeric data type
 Do What I Mean, 396–99
 Drum computers, 8
 Dummy variable. *See* Formal parameter
 DWIM, 396–99
 Dynabook, 404–5
 Dynamic chain, 62, 162. *See also* Dynamic link
 Dynamic link, 62, 219–221, 223–25, 236–38, 433, 435. *See also* Dynamic chain
 Dynamic vs. static, 100, 118–19. *See also* Storage management; Typing
 scoping; Scoping, static vs. dynamic
 Dynamic vs. static structure. *See* Structure Principle
 Dystopians vs. utopians, 33–34, 40, 389
- Economy, 157–58, 244, 303
 Editor
 structure, 397–98, 401–2
 syntax-directed, 248
 Efficiency
 of Ada parameters, 286–88
 and C, 207–8
 of case statement, 201
 and C++, 441
 display procedure call, 233
 display vs. static chain, 234
 display summarized, 234
 display variable access, 231–32
 of enumeration types, 175
 of extensible languages, 169
 fixed-point, 250
 and FORTRAN, 39–40, 57–58
 LISP, 399
 of Pascal parameters, 202
 Prolog, 461, 463–66, 474–75, 480, 490
 of set types, 178–79
 static chain procedure call, 224
 static chain summarized, 230
 static chain variable access, 218
 of subrange types, 176
 Three E's, 157, 303
 Elegance, 156–61
 Elegance Principle, 158–59, 179, 205, 372, 442, 496
 Elementary types, 286
Elements of Style by Strunk and White, 14
 elsif, 302
 Embedded software, 244, 250
 Embodiment, 34
 empty (Prolog), 486–87
 Empty statement, 300–1
 Enabling capabilities, 399
 Encapsulation mechanisms, 163
 end, overused in Algol and Pascal, 301
 Englebart, Douglas, 404
 entry
 Ada concurrency, 293–99
 procedure implementation notation, 64
 Enumeration types, 173–75, 209, 255–56
 Environment (name structure), 41, 78–82, 107–14, 194–96, 258–263, 372, 412–15.
 See also Context; Name structure; Scoping
 of caller vs. of definition. *See* Scoping, static vs. dynamic
 defined, 78
 disjoint, 78
 implementation, 211–218, 231–239, 378, 431–434
 opening, 185
 Environment, program development, 395–99, 443
 Interlisp, 396–99, 443
 programming language as work environment, 160
 Smalltalk, 443
 Environment part or pointer, 212, 214, 221n, 226, 232, 240, 385–86, 433–34
 EP. *See* Environment part or pointer
 EP-IP pair. *See* Closures; Locus of control
 eq (LISP function), 319, 332
 vs. equal, 348
 .EQ. (FORTRAN equality), 46
 equal
 vs. eq, 348
 LISP function, 348–50, 486
 Prolog predicate, 459
 Equality relation, 46, 319, 348–50. *See also* Inequality
 Prolog, 471, 485–87
 Equal symbols. *See also* Symbols at beginning of index
 assignment vs. equality, 99–100, 207
 unification, 471
 Equivalence, type, 191–93, 251–54
 EQUIVALENCE declaration, 84–85, 113
 Equivalence relation, 486–87
 Erasure, automatic vs. explicit. *See* Storage reclamation
 Error correction, automatic 396–99
 Error handlers, 268–69, 284–85, 306–7, 388n
 Euclid (language), 205, 244–45
 Euler (language), 170, 208, 245
 eval (LISP function), 375–88, 395
 evargs (LISP function), 380–81
 Exceptions, 268–69, 284–85, 306–7, 388n
 Exclamation point, 476–80
 Executable statements vs. declarations, 20, 26–27, 41–42, 76. *See also* Declarative vs. imperative; Imperative
 Executable unit, 239–40
 Exit (Prolog trace), 470
 exit-statement, 281–83
 Experience, in design, 14, 136, 160
 Experimental software development, 394–95, 398
 Explicit erasure. *See* Storage reclamation
 expr (LISP property), 333–34, 343, 363–64
 Expression tree, 335, 455–58
 Extended BNF, 153–54
 Extensibility, 163, 168–69, 208, 367, 398–99, 414–15, 417, 423
 Extension, programming by, 436
 Extension of behavior, 414
 Extent of loop, 51
 External references, 44
 External representation, 276–78, 412, 418, 423–24, 436, 441–42
 Extrapolation, amplificatory, 34, 138, 140, 162, 421, 445
 fac (predicate), 476–77
 Facts (Prolog), 447, 449–50
- fail (Prolog)
 predicate, 483
 trace, 470
 Fascination vs. fear, 34–35, 138, 140
 father (predicate), 447
 Fear vs. fascination, 34–35, 138, 140
 Feature interaction, 51, 77, 86–87, 136–37, 158–59, 181–83, 290–91, 302, 304–5
 Featuritis, 304–5, 441
 fib (predicate), 462–69, 471–74
 Fibonacci numbers, 462–69, 471–74
 Fifth-generation, 306, 309, 400–1, 421, 443–44, 497. *See also* Generations
 File system, Interlisp, 398
 filter (LISP function), 352
 Filtering of list, 346–47, 352
 Finite discrete type, 180. *See also* Discrete type
 Finite mapping, 181
 Finite sets. *See* Sets
 First
 array attribute, 255, 282n
 LISP function, 337
 First-class citizen, 70, 116–17, 204, 228, 277, 442
 First-generation language. *See* Generations
 first-order logic, 481
 Fixed vs. free format, 30–31, 86–87, 143–44
 Fixed point of function, 460
 Fixed-point types, 249–50. *See also* Numeric data types
 Flag value, 20
 FLEX, 404
 Flexible arrays, 119
 Float, 249
 Floating-point numbers, 9–10, 36, 39, 66–69, 249
 FLOW-MATIC, 66
 FLPL, 310
 Focus and action, 34, 77, 395
 for-loop, 137–38. *See also* Iteration, definite
 Algol vs. DO-loop, 122
 Algol vs. Pascal while-loop, 122
 baroque, 137–38
 for-list-element, 137
 often unnecessary in LISP, 345–46
 Pascal, 197
 Formal parameter, 54–55, 162, 353–54, 363–64, 382–84. *See also* Parameter passing modes
 Form and function, 13, 159
 Format, fixed vs. free, 30–31, 86–87, 143–44
 FORTRAN, 39–92, 117–23, 146, 161–62, 167, 170, 172–73, 179–81, 189, 197, 202, 211, 220, 249, 268, 286, 305–6, 309–11, 343, 364, 446
 as cockroach, 93
 history, 39–40, 90–91
 lost space probe, 87
 name, 40
 predicted use in year 2000, 93
 Zero-One-Infinity violated, 117–18
 forward declaration, 195–96
 Fourth-generation language, 5n, 305–6. *See also* Generations
 FP. *See* Backus's FP language; Functional programming
 Framework, language, 494–95
 Free-list, 389, 393. *See also* Storage reclamation
 Free vs. fixed format, 30–31, 86–87, 143–44
 Free storage area, 337, 388–394, 428–30, 432–36
 Free union, 251
 Freq (LISP list), 312, 320

- Fully bracketed syntax, 301–2
 funarg, 386n. *See also* Closure; Upward funarg problem
 funcall (LISP function), 356n
 Function. *See also* Procedures
 anonymous, 353–54
 and form, 13, 159
 FORTRAN FUNCTION, 57, 123
 level vs. object level, 359
 LISP function, 356, 370, 385–88
 Pascal, 203
 pseudo-, 316, 339
 pure, 316, 321, 328, 338–39
 as typed procedure, 99
 Functional, 352–57, 359–63. *See also*
 particular functionals
 defined, 355
 Functional arguments. *See* Functional;
 Functional Programming; Parameter
 passing modes, functional
 Functional programming, 309, 355–63,
 400–401. *See also* Functional; Function-
 oriented programming
 Backus's language, 359–63
 defined, 355
 dynamic scoping, 367–70
 implementation, 228
 filtering, 346–47, 352
 mapping, 346, 351–52, 357–58, 360, 427
 in Prolog, 480–81
 reduction, 346, 352, 357–58, 360–61, 480–81
 Function cells (Common LISP), 334n
 Function-oriented programming, 306, 309,
 312–17, 400–401, 421–22, 443–44
 Functor, 451
- Garbage collection, 392–94, 441. *See also*
 Storage reclamation
 Generality. *See* Regularity Principle;
 Simplicity Principle; Zero-One-Infinity
 Principle
 Generations, 5, 92, 138, 163–64, 207–8,
 305–6, 400–401, 443–44, 489. *See also*
 Fifth-generation; Fourth-generation
 language
 reasons for shift, 158
 Generic operators, 68. *See also* Overloading
 Generic packages, 270–76, 423–24, 441
 get (LISP function), 332
 getprop (LISP function), 324–25, 332
 getproplist (LISP function), 334n
 Global, 78–79, 104, 162, 364. *See also* Name
 structure; Scoping; Visibility
 considered harmful, 258–63
 Goals (Prolog), 447–48, 450, 465
 Gödel's incompleteness theorem, 481
 go: message, 407–8
 Go to controversy, 126. *See also* Structured
 programming
 goto statement, 15–16, 45–51, 91, 125–26,
 135–36, 164, 172, 198, 228–30, 281
 nonlocal, 135–36, 228–30
 switch-declaration, 98–99, 139–40
 goto: message, 406–7, 413–15
 GRAIL, 404
 Grammar. *See* BNF
 Grammar size, 303–5
 grandparent (predicate), 447
 Graphics, turtle, 406–10
 Greater-or-equal relation, 177
 Greater-than relation, 48, 174–75, 177
 Green, Cordell, 446
 grow: (message), 410
 .GT. (FORTRAN greater-than), 48
 Guards (guarded entries), 297–98
- Hamming, R. W., xv-xvi, 3
 Handler, exception, 268–69, 284–85, 306–7,
 388n
 Hardware representation, 145
 Hayes, Pay, 446
 Head of Prolog clause, 450, 465
 Heap storage management, 337, 388–394,
 428–30, 432–36
 Heterogeneous data structure, 185–86
 Hidden parameter, 255
 Hierarchical structure, 46–48, 52–53, 89–90,
 97–98, 144, 301–2
 and BNF (context-free) grammar, 155–56
 classification (Smalltalk), 412–15, 418–21,
 443
 LISP equality, 348–50
 nested statements, 122–26
 Higher-Order Language Working Group,
 244–45, 249
 High level. *See* Level, higher vs. lower
 Hilfinger, Paul, 307
 History, 7–11, 36–37, 39–41. *See also*
 individual programming languages
 Santayana on, 3
 History list, 397
 Hoare, C. A. R., 29, 167, 170, 183, 188, 200,
 304–5, 307
 Hollerith, Herman, 69n
 Hollerith constant, 69–70
 HOLWG, 244–45, 249
 Homogeneous data structure, 185–86
 Hopper, Grace Murray, 11, 39
 Horn clause form, 450, 482–83, 489
 Hypotheses (Prolog facts), 447, 449–50, 465
- IAL (International Algorithmic Language),
 95–96, 119, 162, 301
- IBM
 and Algol, 96, 162
 and FORTRAN, 39–40
 360 computer, 249
 650 computer, 8, 11
 701 computer, 39
 704 computer, 9, 36, 39–40, 45–46, 66,
 161, 206, 311, 337
 Ichbiah, Jean, 245
 Identifiers (names), 76, 117–18, 146. *See also*
 Name structures
 Identity of Indiscernibles, 485
 if (LISP function), 324–25, 344, 377, 379
 if (Prolog), 450n
 IF-statements, FORTRAN, 45–49, 122–23
 if-then-else. *See* Conditional selection
 Ihde, Don, 33–35, 404
 Image, system, 135
 Imperative, 31n, 43, 45, 98–100, 345n. *See*
 also Executable statements vs.
 declarations
 language, 313, 339
 programming in Prolog, 471–74, 477–80
 Implementation dependence and
 independence. *See* Machine dependence
 and independence; Portability;
 Portability Principle
 Implementation vs. specification, 257
 Implicit bindings (enumeration types),
 173–75, 193, 255–56
 Implicit inheritance, 104–5, 262
 Importation, 262, 264, 266
 Impossible Error Principle, 12, 52, 58, 106,
 180, 191, 496
 in
 Ada passing mode, 286–88
 Pascal set membership, 177, 179
- Indenting, 124–5, 143–44. *See also*
 Structured programming
 Index constraint, 254–55
 Indexing, 9, 17–18, 36, 39, 53, 73–75,
 180–81. *See also* Arrays
 Index registers, 73–75
 Index type, 180–81
 Indicator, 324
 Indiscriminate access, 107 258–60, 262–63
 Inequality, 482, 485–86, 488–89
 Infinite terms, 459–60. *See also* Lists, circular
 or cyclic
 Infix format, 31, 451, 453, 455–56
 Information hiding, 78, 245, 257, 263–65,
 280, 305, 496. *See also* Information
 Hiding Principle
 Information Hiding Principle, 81, 107, 263,
 408–9, 416, 426, 429, 433, 435, 442. *See*
 also Information hiding
 Ingerman, P. Z., 132
 Inheritance
 class, 413–15, 418–21
 implicit vs. explicit, 104–5, 262
 multiple, 421, 441–42
 Initialization, 28–29, 42, 256
 Inner product, 352, 360–63
 in out (Ada passing mode), 286–88
 Input-output, 18–19, 43, 161, 197, 474
 I-N Rule, 77
 Inspection time, 134. *See also* Binding time
 Instance method, 410
 Instance variable, 409–11, 429–31, 433–34
 Instantiation, 127–28, 240, 271
 object, 408–9, 431
 Prolog, 454, 472
 static vs. dynamic, 272, 277, 423–24
 Instruction decoding, 23–24
 Instruction part or pointer, 21–22, 212, 221n,
 226, 240, 385–86, 388, 433
 Integers, 67, 115. *See also* Numbers; Numeric
 data types
 vs. enumeration types, 173–74
 Prolog definition, 451–52
 Smalltalk, 408
 Integrated programming environment. *See*
 Environment, program development
 Interface, manifest, 316–7, 416, 496
 Interface overhead, 37, 169
 Interface specification, 248, 257, 305, 395,
 398
 Interim Dynabook, 404
 Interlisp, 312, 396–99, 402
 Intermediate form, 32
 Internal representation. *See* Representation,
 internal vs. external
 International Algorithmic Language (IAL),
 95–96, 119, 162, 301
 Interpreter
 efficiency, 11, 23–24, 36–37, 39, 399
 iterative (pseudo-code), 21–33, 375
 origin, 10
 recursive (LISP), 310–11, 375–388
 universal function, 310–11, 375
 Intersection, 177, 179
 Intuition pump, 157
 Invariant code, removing from loop, 74
 Inverse functions, 452, 466–69, 472–74
 Invocation (procedure call), 55. *See also*
 Procedures
 IP. *See* Instruction part or pointer
 ip (function), 352, 360–63
 IP-EP pair, 214, 240
 IPL, 309
 Ironman, 245
 is (Prolog), 471

- ISO (International Standards Organization).
See Standards
- Iteration, 281–84, 376–82
Ada loop, 281–283
Algol **for**-loop, 122
and bottom-up control, 463–65
definite, 17–18, 51–53, 122, 137–38, 197, 282–83, 408
indefinite, 47–49, 198, 426
infinite, 281, 426
leading-decision, 47, 198, 282
mid-decision, 48, 198, 282
optimization, 73–75
Pascal **for**-loop, 197
vs. recursion, 345–47, 350
Smalltalk, 408, 426
trailing-decision, 47, 198
- I Through N Rule, 77
- Iverson, Ken, 372
- Java (programming language), 390, 441
Jensen's device, 110, 131–32, 313
Johnston, John B., 79
JOVIAL, 96
- Kay, Alan, 403–4, 421, 442–44
Kernel language, 168
Kernighan, B., 207, 210
Keywords, 145–46. *See also* Reserved words
Kleene, S. C., 153n
Kleene star and cross, 153, 156
Knuth, Donald, 141, 165
Kowalski, Robert, 446, 461n, 491
Kron, H., 56n
- Label, statement, 26–28, 49–51, 193, 196, 198, 228–30, 240
Labeling Principle, 26, 200–1, 289, 425, 496
Label table, 27–28
LABEL type, 51
lambda (LISP form), 334, 353–55, 385–88
Lambda calculus, 353
Lambda expressions, 354–55, 385–88, 427
Landin, Peter, 126
Languages, 1–2. *See also* Programming language
formal, classes, 154–56
object vs. metalanguage 150–51
reference vs. publication, 145
Laning and Zierler system, 39–40, 89
Large, programming in the, 56, 357–58
Last, array attribute, 255, 282n
Leading-decision loop, 47, 198, 282
left (field), 335, 36, 300, 307
length (LISP function), 327, 359
Lenient interpretation of logical connectives, 344–45, 349–50, 379
Less-greater symbol (<>), 177. *See also* Angle brackets
Less-or-equal relation, 177, 179
Less-than relation, 48, 174–75, 177
let (LISP function), 366–67
Level, higher vs. lower, 10, 36, 51–52, 175, 177, 188–89, 351, 359, 441, 445–46, 490
Lexics and lexical analysis, 30–33, 85–88, 143–46
Libraries, 11, 36–37, 44, 56, 267
Library items, 266
LIFO (last-in, first-out), 113–14, 436
Linear structure, 89–90, 92. *See also* Hierarchical structure
Linking, 43–44
lint, 207
Lisa computer, 405
- LISP, 250, 302, 309–401, 405, 411–2, 422, 436, 441–43, 446, 452–55, 480–81, 490.
See also Common LISP
dynamic scoping, 107, 112, 365–70, 385–88
history, 309–12, 396, 399–400, 441–42
LISP 2 language, 311, 370, 400
longest still in use except FORTRAN, 311
“Lots of Idiomatic Single Parentheses,” 371
machines, 399, 402
programs represented by lists, 315, 371–2, 395
- list
LISP function, 347, 354
Prolog predicate, 453
- Lists, 309–11, 314–16, 319–40, 452–56
association, 326–27, 378
cdr-encoding, 342
cell, 335
circular or cyclic, 340, 391–92, 429
editor, 397–98, 401–2
implementation, 334–36
left and right parts, 335
null (empty), 319
Prolog implementation, 452–55
property, 324–26
recursive construction, 328–30
represent programs, 315, 371–2
shared sublists, 338–40
- Literals
numeric, 71, 148–54
Prolog, 450
- Literal table, 59, 128
Loaders and loading, 19–20, 24, 27–28, 30, 32, 43–44
Local, 78, 104, 162, 198, 382–84, 411. *See also* Name structure; Scoping; Visibility
in activation record, 213
multiple instantiation of local variables, 127–28, 213
- Localized Cost Principle, 138, 206, 290, 495–96
Locus of control, 214, 240
Logic, first-order vs. higher-order, 481
LOGICAL (FORTRAN data type), 66
Logical IF-statement
Logic vs. control, 461, 469, 473–74, 480, 489–91
Logic-oriented programming. *See* Logic programming; Prolog
Logic programming, 306, 309, 445–51, 461–80, 489–90. *See also* Logic vs. control
concurrency, 465–66
vs. logic-oriented programming, 483, 485, 489n, 469–74, 477–80, 483–85, 487–90
procedural interpretation, 465–69
- LOGO, 404, 406
Long_Float, 249
Loopholes. *See* Security Principle
Loops. *See* Iteration
loop-statement (Ada), 281–83
Lovlace, Countess of (Augusta Ada), 245
Lower case, 31n, 450
.LT. (FORTRAN less-than), 48
Łukasiewicz, Jan, 313
L₁ and L₂ (pseudo-codes), 11
- M (contents of memory location), 63
Machine dependence and independence, 45–46, 70, 72, 95, 115, 120, 143–46, 249, 441
Machine-oriented higher-order language, 45–46, 188–89, 205–8. *See also* Portability Principle
- Macintosh computer, 405
MacLennan, B. J., 305n, 350n, 353n, 355n, 444n
MacLisp, 312
Macros
Interlisp, 398
syntax, 168–69
MAD language, 168
Mailbox, 294
Maintenance, program, 26, 59, 81, 106–7, 123–25, 193–4, 200, 243, 263–5, 271, 300–1, 398, 459
Manifest Interface Principle, 316–7, 416, 496
Mantissa, 67
map (LISP functional), 357, 360
mapcar and mapcar2 (LISP functions), 351–52
Mapping, finite (array), 181
Mapping across a list, 346, 351–52, 357–58, 360, 427
map2 (LISP functional), 358
Mariner I, lost, 87
Mark bit, 392–94
Mark-sweep garbage collector, 392–94
Masinter, Larry, 396n, 402
Masterscope, 398
Matching. *See* Unification
mat-prod (function), 361–63
Matrix multiplication, 361–63
McCarthy, John, 309–11, 337, 344
McIlroy's syntax macros, 168
McLuhan, Marshall, 404
Mean absolute value program, 20, 32, 42, 98, 171
Meek, Brian, 91n
member (predicate), 486
Mesa, 205, 244
Message dictionary, 431–32, 434
Message port, 294
Message sending, 294, 406, 414–18, 423–27, 434–36, 443
Messages, Smalltalk, 406–8, 423–26
cascaded, 408
implementation, 434–36
templates, 424–26, 431–32
Metalanguage, 150–51
Method (Smalltalk)
class, 410–11
implementation, 431–32
instance, 410
as procedure, 423–25
Metric, grammar size, 303–5
M-expressions, 311
minusp (LISP function), 366
minusp-fil (LISP function), 346–47
Minus sign (-), 177
double (Ada comment), 265
Mission critical software, 244, 250
Mixed-mode expressions, 69
Models, 134–35, 411–12, 458–59
Modes, parameter passing. *See* Parameter passing modes
Modula, 205, 208, 244
Modularization, 55–56, 243–44, 248, 263, 438. *See also* Information hiding; Information Hiding Principle; Modules; Packages; Procedures
Modules. *See* Information hiding; Information Hiding Principle; Modularization; Packages; Procedures
MOHOL (Machine-oriented higher-order language), 45–46, 188–89, 205–8
Monitor. *See* Protected type
Monotonic reasoning, 475

- Montessori, Maria, 404
month (data type), 174, 184
mother (predicate), 447
Multiple inheritance, 421, 441–42
Multiplication, matrix, 361–63
- Name access, 262, 264, 266
Name equivalence, 191–92, 251–53. *See also*
Type equivalence
Name parameters, 129–35, 162, 201, 222, 240
Name space. *See* Name structure
Name structure, 75–76, 92, 101–4, 163, 193,
205, 256, 363–64
clutter, 353
contour diagrams. *See* Contour diagrams
records as, 185
Naur, Peter, 96, 126, 148, 163
Negation, 481–85
NELIAC, 96
Nesting. *See* Hierarchical structure; Scoping
Neumann, John von, 8, 9
new
Ada generic instantiation, 271
Ada storage allocation, 269
Ada type declaration, 251–54, 438
Pascal storage allocation, 189–90, 263,
337, 388–89
Smalltalk instantiation, 411
newAt : (message), 409, 411
nil (atom), 319, 325
Prolog definition, 452–55
n1 (predicate), 478
no (Prolog response), 448
Nonexecutable statements. *See* Executable
statements vs. declarations
Nonlocal, 104. *See also* Display; Local; Static
chain
nonlocal access, 213–15, 382–88, 434
Nonmonotonic reasoning, 475–76
Nonnumeric data, 69–70, 172–75, 314–15,
318–19. *See also specific data types*
Nonprocedural programming, 445–46, 459,
469, 473–74, 479, 490
Nonterminal symbols, 150
Norman, Donald, 134–35
not (predicate), 483–85, 488–89
Notation, 148, 362, 372. *See also* Symbols at
beginning of index
Not-equal, 482, 485–86, 488–89
NPL (New Programming Language), 90. *See*
also PL/I
null (LISP function), 319
Prolog definition, 452–53
numberp (LISP function), 377
Numbers. *See also* Decimal numbers;
Numeric data types
boldface, 362
denotations, 148–54
Numeric data types, 66–70, 115, 248–50,
252–54, 256, 317–18. *See also specific*
data types
- Oberon (language), 205
Object (object-oriented programming), 404–9,
421–22, 426, 429–30
object class, 413, 431, 434
self, 411
self-displaying, 416–17
Object code, 43–44
Object declaration (Ada), 256
Objectification of tools, 34
Object language, 150–51
Object-level vs. function-level, 359
Object-oriented languages. *See* Object-
oriented programming
- Object-oriented programming, 163, 277–78,
303, 306, 309, 399, 443–44, 458–59
in Ada 95, 436–41
in C++, 441
in Java, 441
in LISP, 441–42
as simulation, 411–12, 427, 443–44, 486
used loosely, 443n
Object pointer or reference, 422, 433
Occurs check, 460
Offset, relative, 216–7, 240, 433
onep (LISP function), 318
Operator extension, 168–69, 264–65, 278–79,
415–18. *See also* Extensibility
Operator identification, 278–79, 290–91
Operator overloading. *See* Overloading
Optimal coding (drum computers), 8
Optimization, 7–8, 40, 44, 53, 73–75, 248,
399
Optional parameter, 279, 289–91
Optional syntax, 154
or (LISP function), 344–45
Ordinal type, 176. *See also* Boolean type and
values; char; Enumeration type;
Subrange type
Orthogonality Principle, 13–15, 124, 136,
203, 263, 286–87, 418, 420, 442–43,
461, 489, 495–96
otherwise, 210
out (Ada passing mode), 286–88
Overlapping definitions, 258, 261–63, 267–68
Overloading, 68–69, 248, 255–56, 264, 274,
278–79, 290–91, 415–16, 441
own variables, 113n, 140, 263
- Packages, 248, 264–79, 436–40. *See also*
Class
generic, 270–76, 423–24, 441
Pair, dotted, 326n, 453–55
pairlis (LISP function), 353, 384
Papert, Seymour, 404
PAR (part of activation record), 63, 221
Paradigm, 412, 443–44
Parallelism. *See* Concurrency
Parameter
actual, 55, 162, 222, 227
default, 279, 289–91
formal. *See* Formal parameter
hidden, 255
optional, 279, 289–91
position-independent, 279, 288–91, 425
Parameter passing modes, 286–88
constant, 202–3, 286–88
functional, 203–4, 225–28, 240, 351–63,
367–70, 385–88
implementation, 222, 274–76
input, 202–3, 286–88
input-output, 202–3, 286–88, 466–69
input-output the same in Prolog, 466–69
instantiation. *See* Instantiation
internal vs. external representation. *See*
Representation, internal vs. external
Jensen's device, 110, 131–32, 313
literal table corrupted, 59
name, 129–35, 162, 201, 222, 240
output, 286–88
parameter inspection time, 134
procedural. *See* Parameter passing modes,
functional
reference, 56–60, 134, 201–2, 222, 286–88
result, 286–88
swap procedure impossible in Algol-60,
132–34
test procedures, 60–61, 130–31, 141
think, 132, 140, 162, 202, 222
- typed vs. untyped, 99
value, 128–29, 134, 162, 201–2, 222
value-result, 60–61, 286–88
PARC, Xerox, 396, 399, 404
parent (predicate), 447
Parentheses (LISP), 314–15, 319–20, 370–72
Parnas, D. L., 81, 243, 280
Parnas's Principles, 263. *See also* Information
hiding; Information Hiding Principle
Parser, 44, 86, 155, 248
Pascal, 117, 119, 138–39, 163, 170–205, 211,
228, 235, 237, 245, 247–49, 251,
254–55, 257, 263, 281–82, 286, 289,
299–301, 303–6, 343–46, 372, 389, 395,
422, 446, 463–64, 480. *See also* P-code
concurrent, 205, 306
goals, 170
history, 170–1
Passive vs. active, 421–22
Path, access, 185
Path expression, 306
pcc, 207
P-code, 36, 170–71
PDP computers, 206–7
pen (Smalltalk class), 406–10
pendn and penup (messages), 406–7
Period
double (. .), 175, 252
single (.), 184, 266, 269, 272, 274, 453–55
Perlis, Alan, 96–97, 162–63
person (record type), 184
Personal computer, 403–5
Phenomenology, 33–35, 404
programming language as work
environment, 160
technology is nonneutral, 33, 35, 158
values symbolized by aesthetics, 160
Piaget, Jean, 404
plane (record type), 187
Plato, 404
PL/I, 69, 90–91, 117, 146–47, 162, 164–165,
167–68, 189, 205–7, 249
complexity criticized, 90, 167–68, 304
as fatal disease, 90
as plane, 167–68
as Swiss army knife, 167
plist (LISP function), 334n
p-list, 324–26
PL/S, 206
plus
LISP function, 317
Prolog functor, 451, 455–57
plus-red (LISP function), 346
Plus reduction. *See* Reduction of a list
Plus-sign
Kleene cross, 153
set union, 177
PL360, 170, 208
pname (LISP property), 331
Pointer, dangling, 292–93, 389
Pointer binding, 190
Pointers. *See* Pointer types; References and
referencing
Pointer types, 189–91, 250, 440
Polish notation, 313
Polymorphic, operations, 68. *See also*
Generic packages; Overloading
Portability, 95, 143–46, 207–8, 303, 441, 459.
See also Machine dependence and
independence; Portability Principle
Portability Principle, 46, 115, 143, 246, 287,
429, 496
Position-independent parameter, 279, 288–91,
425
Postfix format, 31, 456n

- Precedence of operator, 89, 168, 313, 425–26
Precision, numeric. *See* Numeric data types
pred (Pascal function), 174–75
Predicate (Prolog), 450
 input-output, 474
Predicate logic, 481
Predication (Prolog), 450
Predictability, 135. *See also* Regularity;
 Regularity Principle
Prefix format, 31, 456. *See also* Polish
 notation
Preprocessors, 91
Preservation of Information Principle, 53,
 188–89, 249, 496
Primitives vs. constructors, 70, 75
Principles of programming language design,
 3, 14, 495–96. *See also specific*
 principles
 applying, 14, 495–96
 collected, 496
print (message), 416–17
Print name, 331–32
Priority of operator, 89, 168, 313, 425–26
Private
 part of package specification, 264–65
 type, 264–65, 273–74, 437–38
Procedural abstraction. *See* Procedures
Procedural parameters. *See* Parameter passing
 modes, functional
Procedural programming. *See* Nonprocedural
 programming
Procedures, 54–56, 127–34, 286–91, 423–26,
 465–69. *See also* Declarations,
 procedure; Message sending; Parameter
 passing modes
 Ada, 247, 257, 264, 267–79, 286–91, 300
 Algol, 99, 127–34, 195
 anonymous. *See* Anonymous
 array parameters, 57–58, 128–29, 182–83
 as first-class citizens, 204
 FORTRAN, 54–56
 implementation, 61–66, 219–35, 239–40,
 380–88, 432–36
 implementation notation, 63–64
 LISP, 317, 333–334, 351–57, 363–64,
 367–71, 380–88
 Pascal, 172, 195
 Prolog, 465–69
 recursive, 127–28, 195–96
 Smalltalk. *See* Messages, Smalltalk;
 Method (Smalltalk)
 specification, 257
 Process, 240, 248, 291–99, 427–28
 prod (predicate), 457
 Product, inner or scalar, 352, 360–63
 prog feature (LISP), 345n
 Program vs. data, 9–10, 315–16, 421–22
 Program design notation, 8
 Programmer's Assistant, 397–99
Programming
 difficulty of, 7
 experimental, 394–95, 398
 by extension, 436
 in the large, 56, 357–58
 reliable, 111–12
 software crisis, 243
Programming environment. *See* Environment,
 program development
Programming languages, 1–2. *See also*
 specific languages
 applicative vs. imperative, 313
 design, vs. feature design, 305
 featuritis, 304–5
 framework, 494–95
 future of, 497
 number of, 4, 244
 perfect, 493
 phenomenology. *See* Phenomenology
 reason for diversity of, 493–94
 size, 97
 small programs misleading, 5
 and values, 160
 as work environment, 160
Programming in the large, 56
Programming, nonprocedural. *See*
 Nonprocedural programming
Programming, structured, 90–91, 125–26,
 164, 195–97
Program structure, 19–20, 31, 41, 97–98, 196
Prolog, 445–90. *See also* Equality; Inequality;
 Logic programming; Negation
 cuts, 476–80, 490
 database, 447, 458, 475–76
 depth-first search, 469–71
 history, 446
 logic vs. control, 461, 469, 473–74, 480,
 489–91
 order of clauses, 450, 461
 Propagation of exceptions, 284–85
 Properties, of atoms. *See* Atoms, properties
 Property lists, 324–26
 Protected type, 297–99
 Protocol, class, 412, 416–17, 422
 Proving, automatic, 446, 459–60, 466, 469,
 475–76
Pseudo-codes, 7–33
 defined, 10
 operations (table), 19
 Smalltalk, 429, 432
Pseudo-function, 316, 339
Psychology, 134–35, 404
Publication language, 145
Pure function, 316, 321, 328, 338–39
putprop (LISP pseudo-function), 332, 391n

Quotation marks, single, 21n, 315, 351, 371.
 See also quote (LISP)
quote (LISP), 371, 377, 379

raise, 269, 284–85
Range
 array attribute, 255
 constraint, 248–50, 254
Range constraint, 248–50, 254. *See also*
 Subrange type
RATFOR, 91
read (predicate), 474
Readability, 47–50, 86, 116, 118, 143–46,
 148–50, 153–54, 193–94, 271, 279, 291,
 301–2, 323, 343
Read-execute cycle, 22, 316–17, 426–27
Real numbers, 9–10, 36, 39, 66–69, 249
real type (Algol), 98, 115. *See also* Real
 numbers
Records, 183–88, 251–52, 315, 455
Recursive definition, 127, 151–52, 310, 447,
 452, 454. *See also* Procedures, recursive
 base of, 127, 151–2
 and hierarchical structure, 348–50
 indirect, 155–56
 and induction, 329
 vs. iteration, 345–47, 350
 mutual, 195–96
 and top-down control, 461–63
Recursively enumerable grammar and
 language, 154
red
 LISP functional, 358, 360
 Prolog predicate, 480–81
redo (Programmer's Assistant), 397
reduce (LISP function), 352
Reduction of list, 346, 352, 357–58, 360–61,
 480–81
Reductive vs. ampliative, 33–34, 162, 316,
 396, 421, 480
Reference, dangling, 292–93, 389
Reference, object, 422
Reference, pass by, 56–60, 134, 201–2, 222,
 286–88
Reference counts, 390–92, 394, 429, 435. *See*
 also Storage reclamation
Reference language, 145
References and referencing, 28, 57, 390, 422.
 See also Pointer types
Refinement of classes, 413, 438, 440
Regular grammar and language, 154–56
Regularity Principle, 10–11, 13–16, 36, 70,
 71, 101, 116, 117, 121–23, 136, 257,
 285, 328, 408–9, 421, 425, 431, 442,
 452, 495–96
Relationship (Prolog), 450
Relative offset, 216–7, 240, 433
Relocation and relocatable format, 43–44
remassoc (LISP function), 330
Rendezvous, 293–99, 427
repeat (predicate), 477–79
repeat-until, 47, 198. *See also* Iteration,
 indefinite
Representation, internal vs. external, 276–78,
 412, 418, 423–24, 436, 441–42
Representation consistency, 398
Representation independence, 67, 120, 244,
 268, 459. *See also* Machine dependence
 and independence; Portability Principle
Resatisfaction, 475
Reserved words, 88, 146, 171
Resolution algorithm, 460, 469, 481
Responsible Design Principle, 114–15,
 188–89, 389, 496
rest (LISP function), 337
Restrictions. *See* Regularity Principle;
 Simplicity Principle; Zero-One-Infinity
 Principle
Resumption address, 63–64, 220–21. *See also*
 Instruction part or pointer
Retention vs. deletion, 240, 293n
retract (predicate), 475, 478–79
Return, implementation. *See* Procedures,
 implementation
RETURN-statement, 54–55, 225. *See also*
 Procedures, implementation
rev (LISP functional), 357
Reversing arguments, 357
Richards, Martin, 206
right (field), 335–36, 390, 397
Ritchie, Dennis M., 206–8
Robinson, J. Alan, 469, 481
roots-aux (function), 365–67
Roussel, Philippe, 446
Row-major order, 72
rplaca and rplacd, 339–40, 390–92, 397
Rules (Prolog), 450
run message, 426–28
Run-time vs. compile-time, 11, 36–37, 41–44,
 100, 215–18
Ryle, Gilbert, 495

Safety features. *See* Security; Security
 Principle
Sammet, Jean, 40
Sandewall, Eric, 396n, 402
Santayana, George, 3
Sapir-Whorf hypothesis, 2
Satisfaction of goal (Prolog), 448, 481–85
Scalar product, 352, 360–63

- Scaling, manual, 9–10
Scanner. *See* Lexics and lexical analysis
Scheduling, Smalltalk, 427–28
Scheme (programming language), 370, 388, 400
Scientific dimension (three S's), 157
Scope defining constructs, 212. *See also* Name structure; Scoping
Scoping, 162. *See also* Context; Environment (name structure); Name structure; Visibility
 global, 78–79
 lines, 102
 local, 78
 records, 185
 static vs. dynamic, 107–12, 219–20, 284–85, 365–70, 385–88, 400
Scribe (Smalltalk object), 406–8
sd, 216
Second-class citizen. *See* First-class citizen
Second-generation language. *See* Generations
Security, 12, 36, 113, 119–20, 173–76, 179, 186, 189–92, 200, 203–4, 207, 389–90, 441. *See also* Security Principle
Security Principle, 27, 29, 59–60, 70, 77, 202, 419–20, 422, 442, 459, 495–96. *See also* Security
Selection, component. *See* Selector
Selection statements, 47. *See also* Case selection; Conditional selection
Selector, 185–86, 320, 327–29
 implicit, 454–55
 select-statement, 296
 self (Smalltalk), 411, 414–15
Self-documenting, 200–1, 489
Self-embedding terms. *See* Infinite terms
Self-referential structures. *See* Lists, circular or cyclic
Semantics, 97, 163, 248
Semicolon
 Prolog, 448, 468, 484
 terminating vs. separating, 300–1, 307
Sender part (Smalltalk), 433. *See also* Dynamic link
Separating semicolon, 300–1
Sequential interpretation of logical connectives, 344–45, 349–50, 379
set (LISP pseudo-function), 316–17, 332–33, 363–64, 371, 391n
set_equal (predicate), 487
setq (LISP pseudo-function), 371
Sets
 Pascal, 176–79
 Prolog, 486–87
sex (data type), 174
S-expressions, 311–12, 315, 319, 370
sg (signum or sign function), 343–44
Shallow binding method, 234
Shared access, 80–84, 102–7, 112–14, 261–3, 267–68, 338–40
Shaw, Mary, 258, 262–63, 307
Short_Float, 249
sibling (predicate), 447
Side effects, 59, 258–59, 262–63, 316
signum (sign function), 343–44
Simplicity, metric, 303–5
Simplicity Principle, 116, 136–37, 168, 197, 205–6, 208, 257, 314–5, 421, 442, 452, 495–96
Simula, 163, 244, 272, 277, 404, 412, 427, 436, 441, 444
Simulation, 411–12, 427, 443–44, 458–59
Size, language metric, 303–5
Sketchpad, 404
SL (part of activation record). *See* Static link
Slash
 symbols. *See* Symbols at beginning of index through list cell, 335
Smalltalk, 163, 272, 277, 399, 403–444. *See also* Class; Messages, Smalltalk; Method (Smalltalk); Object (object-oriented programming)
 history, 403–5, 444
 implementation, 428–35
 read-execute loop, 426–27
snl, 216
Social dimension (three S's), 157–58
Software, experimental, 394–95, 398. *See also* Programming
Software crisis, 243
Source form, 32
Space probe lost, 87
Special names (LISP), 388
Special needs annexes (Ada), 246, 303
Specification, 248, 257, 305, 395, 398
Speedcoding, 39
Stack
 example, 268–78, 416
 runtime storage management, 100, 112–14, 118–19, 211–40, 435–36
Standards, programming language
 Ada, 245
 FORTRAN, 40, 91–93
 LISP, 312, 399–400, 441–42
 Pascal, 170, 182–83, 191–92, 195–96, 198, 204
 PL/I, 165
 Scheme, 400
 Smalltalk, 404
Star, Kleene, 153, 156
State, of caller, 61–62, 218–21, 434–35
Statement, empty, 300–301
Static chain, 162, 214–15, 234, 434. *See also* Static link
 Static distance, 216
 Static vs. dynamic. *See* Dynamic vs. static
 Static vs. dynamic structure, 111. *See also* Structure Principle
 Static link, 213–15, 219–230, 236–38. *See also* Static chain
 Static nesting level, 216
 Steelman, 245
 Storage allocation. *See* Storage management
 Storage management, 28–30, 42, 100, 112–14, 118–19, 127–28, 196, 211–40, 388–394, 428–30, 432–36
 FORTRAN EQUIVALENCE, 84–85
 Storage reclamation, 388–94, 399, 429. *See also* Storage management
 Strawman, 245
 Strict interpretation of logical connectives, 344–45, 349–50, 379
 string (type), 116–17, 184
 Strings, 69–70, 116–17, 173, 319, 415–18. *See also* Atoms
 Strong typing, 119–21, 190. *See also* Weak typing
 Stroustrup, Bjarne, 441
 Structural engineering, 156–60
 Structural equivalence, 191–92, 252. *See also* Type equivalence
 Structure, hierarchical. *See* Hierarchical structure
 Structured assembler, 206
 Structured programming, 90–91, 125–26, 164, 195–97. *See also* Structure Principle
 Structure Principle, 48–49, 53, 91, 112, 126, 144, 285, 302, 496. *See also* Structured programming
Structures. *See also* Primitives vs. constructors
 heterogeneous data structures, 183–88, 251–52, 315, 455
 LISP, 232n
 Strunk and White, *Elements of Style*, 14
 Style, influenced by tools, 34–35. *See also* Conventions; Strunk and White, *Elements of Style*
 Subclass, 412–15, 418–21, 437
 Subgoal generation, 461–65, 469–71
 Subprograms. *See* Procedures
 Subrange type, 175–76, 248–50, 252–55. *See also* Range constraint
 Subroutines. *See* Procedures
 Subset, language, 246, 249, 303
 Substitution, 56–57, 129–35, 140
 Subten symbol, 148
 Subtype, 251–54
 sub1 (LISP function), 317–18
succ
 Pascal function, 174–75
 Prolog functor, 451–52
Sugar, syntactic, 71, 181, 367, 453–54, 456
sum (predicate), 451–52, 457–58, 471
Summation function, 109–10, 131–32, 254–55
sup (functor), 455
super (Smalltalk), 415
Superclass, 413, 418–21, 431
Superset, language, 246, 249, 303
Suspending. *See* Activation and deactivation, procedure
Swap procedure, impossible in Algol-60, 132–34
Sweep phase, 392–94
Swiss army knife, 167
switch-declaration, 98–99, 139–40
Symbol
 defined, 28
 miscellaneous. *See* Symbols at beginning of index
 terminal vs. nonterminal, 150
Symbolic differentiation, 451, 455–58
Symbolic dimension (three S's), 157–59
symbol-plist (LISP function), 334
Symbol table, 28–29, 76–77, 215–16. *See also* Label table
 shared access example, 80–84, 105–7
Symmetry. *See* Orthogonality Principle; Regularity Principle
Synchronization, 293–99, 427
Syntactic analysis (compilation), 44, 86, 155, 248
Syntactic Consistency Principle, 50, 300, 302, 417–18, 496
Syntactic structure. *See* Syntax
Syntactic sugar, 71, 181, 367, 453–54, 456
Syntax, 30–33, 88–90, 92, 97, 146–47, 164, 299–302, 370–72, 395, 424–26
 two-dimensional, 409
 See also BNF; Extensibility; Lexics
Syntax-directed editor, 248
Syntax macros, 168–69
System image, 135
Systems implementation language, 45–46, 188–89, 205–8
Systems programming, 120, 189, 205–8, 428–29. *See also* Systems implementation language
t (LISP atom), 316
Table. *See* Label table; Literal table; Symbol table
Tacoma Narrows Bridge, 158–59

- Tag field, 187, 251
 Tagged type, 436–40
 tail (function), 360
 Tartan (programming language), 244, 307
 Tasks, 240, 248, 291–99, 427–28
 Teaching, languages for, 170–71, 205, 400
 Technology. *See* Phenomenology
 Teitelman, Warren, 396n, 402
 Template
 C++, 441
 generic package as, 271
 protected type as, 298
 Temporal reasoning, 475–76
 Temporaries, 61, 63–64
 Temporary bindings, 364–67
 Temporary variable (Smalltalk), 411, 433–34
 Ten, subten symbol, 148
 Term (Prolog), 450–55. *See also* Infinite terms
 Term equality, 471, 485–87
 Terminal symbols, 150
 Terminating semicolon, 300–1
 Theorem proving, automatic, 446, 459–60, 466, 469, 475–76
 Third-generation language. *See* Generations
 Thompson, Ken, 206
 Three E's, 157–59
 Three S's, 157–59
 Thunk, 132, 140, 162, 202, 222
 Time, in programming languages, 400–1, 443–44, 475–76, 480
 time (record type), 187
 times
 LISP funtion, 311, 318
 Prolog functor, 455–57
 timesRepeat (message), 408
 Times sign (code duplication), 218
 Tinman, 245
 TMP (part of activation record), 63
 Tokens, 87
 Tools
 descriptive, 148–50
 phenomenology. *See* Phenomenology programming. *See* Environment, program development
 Top-down control, 461–63
 Top-down design, 195
Tower and the Bridge, The, 157–60
 Tracing, 25–26
 Trade-offs, 302–3
 Trailing-decision loop, 47, 198
 trans (transpose function), 360
 Translation. *See* Compilation
 Transparency, 34, 174, 187, 203, 339, 389, 394, 396, 398, 443, 460, 473, 485, 489
 Tree, expression, 335, 455–58
 Truth values, 66, 98, 115, 319
 Turing Award, 167–68, 304, 307, 359, 363, 372
 Turing machine, 1n, 311
 turn: message, 407, 410
 Turtle graphics, 406. *See also* pen
 twice (functional), 367–69
 Two-coordinate addressing, 215–17, 240, 433
 type. *See* Type declaration
 Type conversions, 68–69, 120, 250, 255. *See also* Coercions
 Type declaration, 172, 191–3, 196, 248–54
 Type equivalence, 191–93, 251–54
 Type, protected, 297–99
 Type, tagged, 436–40
 Type 0, 1, 2, 3 grammars and languages, 154–55
 Typing. *See also specific data types*; Type equivalence
 loopholes, 70, 83–85, 187–88, 190
 static vs. dynamic, 100, 181–83, 186, 335, 394–95, 422–23, 440–42
 strong vs. weak, 50–51, 69–70, 119–21, 181–83, 204, 206–7, 422, 441
 typeless languages, 206
 Unary, 3n, 356, 472
 Unconstrained type, 254–55, 440. *See also* Constraint
 Undefined symbols, 27–29
 Undiscriminated union, 251
 undo (Programmer's Assistant), 397
 Unification, 454, 459–60, 462, 486, 488–89
 Union operation, 177, 179
 Union types, 251. *See also* Variant records
 Universal function, 310–11, 375
 Universal integer and real types, 256
 Universal Turing machine, 311
 Unix, 206–8, 304
 Unsatisfiability, 481–85, 488–89
 Upper and lower case, 31n, 450
 Upward compatibility, 305
 Upward funarg problem, 228, 372–73
 use command (Programmer's Assistant), 397
 use declaration, 266, 268–69, 274
 Users, don't ask what they want, 114–15, 188–89, 389, 496
 User's model, 135
 User task, 426–27
 Utopians vs. dystopians, 33–34, 40, 389
 Validation, compiler, 246
 Value, pass by, 128–29, 134, 162, 201–2, 222
 Value cell (Common LISP), 332n
 Value-oriented programming, 444n. *See also* Functional programming; Function-oriented programming
 Value-result, 60–61, 286–88
 Values (desirable traits), represented by languages, 160
 value specification, 128. *See also* Value, pass by
 var, 172
 Variable-free programming, 359–60, 363
 Variables
 accessing, by display, 231–32
 accessing, by shallow binding, 234
 accessing, by static chain, 215–18
 controlled, 51
 declarations. *See* Declarations, variable dummy. *See* Formal parameter instance, 409–11, 429–31, 433–34
 undefined, 27–29
 Variant records, 186–88, 254, 300. *See also* Discriminant; Union types
 Venus space probe lost, 87
 Verification, 245
 Vertical bar, 151, 411, 454n
 Viking (actually, Mariner I) lost, 87
 Virtual computer, 4, 10–11, 36, 428–29
 Virtual procedure, 441. *See also* Abstract subprogram
 Visibility, 78, 104, 135–36. *See also* Context; Contour diagrams; Name structures
 von Neumann, John, 8, 9
 Vulnerability, 111, 258, 260–63
 Weak typing, 50–51, 70, 100. *See also* Strong typing
 when (Ada), 282–83, 285, 297–98
 while-loop, 47, 198, 282, 345. *See also* Iteration, indefinite
 Wilkes, Wheeler, and Gill book, 10
 Window-oriented display management, 405, 409, 443
 Wirth, Niklaus, 114–15, 170, 188, 205, 209, 305
 with-statement, 185, 266–69
 Woodenman, 245
 Word-at-a-time programming, 359
 Work environment, 160
 write (predicate), 474, 478–79
 Wulf, Bill, 258, 262–63, 307
 Xerox PARC (Palo Alto Research Center), 396, 399, 404
 XPL, 206
 Y. *See* Infinite terms
 yes (Prolog response), 468
 Zero-One-Infinity Principle, 71, 117–19, 122, 146, 319, 346, 360, 413, 425, 495–96
 zerop (LISP function), 318
 zerop-map (LISP function), 348, 351
 Zones, 430
 Zuse, Konrad, 9