This appendix lists the languages mentioned in the text, along with information you may find helpful if you want to investigate further. Many of the language names are registered trademarks.

When I say a language is “based on” another, I mean to say that it is in the same general family, even though it may have evolved a great distance from its forebear. Many languages include features from disparate language families and are therefore difficult to categorize. Some are clearly extensions or hybrids of other languages.

Whenever I can, I provide not only pointers to the literature but also URLs (universal resource locators) for getting more information via WWW (the World-Wide Web). These pointers direct you to documentation, examples, compilers, and other language-related information. Several URLs of general interest are `http://union.ncsa.uiuc.edu/HyperNews/get/computing/lang-list.html`, which lists many languages with pointers to more information for each, and `http://cuiwww.unige.ch/langlist`, which lets you interactively search for particular languages. Unfortunately, the WWW changes constantly, so the pointers I provide here may not be valid when you try them.

In a few cases, I describe the syntax and some helpful routines in the language so that you can write small programs and run them.

ABC. Small, interactive, strong typing, indentation for grouping, strings, and exact arithmetic. In use[1]. `http://www.cwi.nl/~guido/ftp/steven/www/abc.html`.

Ada. Large, imperative, compiled, strong typing, concurrent; based on Pascal. In slowly increasing use. Has an ANSI standard. A revision called Ada 95 was published in 1995 (ISO 8652:1995); it includes object orientation[2]. `http://lglwww.epfl.ch/Ada/9X/9X.html`.

AL. Imperative, control of a robot arm. Experimental; in use during the 1970s and 1980s at Stanford University[3].

ALBA. Object-oriented, concurrent. Experimental[4].

ALF. Multiparadigm: object-oriented and logic; based on Smalltalk. Experimental[5]. `ftp://ftp.germany.eu.net/pub/programming/languages/LogicFunctional`.

Algol. Imperative, static types, modern control structures. Pioneered free format, compound statements, variables declared with type, recursion, value-mode parameters. Hoare says that Algol-60 was ‘‘a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors’’[6]. In use in the 1960s, particularly in Europe. Algol-68, Algol-W, and Jovial are independent developments that grew out of Algol-60; they were in moderate use in the 1960s and 1970s[7, 8].

Alphard. Strongly typed, imperative, pre- and postconditions for procedures. Experimental[9].

Amber. Strongly typed, dynamic and static typing, structural equivalence. Experimental[10].

APL. Matrices, interpreted. In widespread but sparse use since the 1960s[11, 12, 13]. `http://www.acm.org/sigapl`.

Argus. Imperative, concurrent, strongly typed, compiled, transactions; based on CLU. Experimental[14, 15].

Awk. Strings, interpreted. In widespread use, particularly on Unix[16]. Available in the GNU software suite as `gawk`. `ftp://netlib.att.com/research/awk*`.

C. Imperative, systems programming; based on Algol. In heavy and increasing use since the 1970s. Has an ANSI standard[17, 18]. `http://www.cis.ohio-state.edu/hypertext/faq/usenet/C-faq/top.html`. Available in the GNU software suite and ported to a great number of platforms.

C++. Object-oriented, extends C. In heavy use[19, 20]. Available in the GNU software suite. For MS-DOS, a nice and inexpensive implementation is available from Borland International, 1800 Green Hills Road, Box 660001, Scotts Valley, CA 95067.

Canopy. Concurrent, extends C. In use at Fermilab in Illinois[21].

Charm. Concurrent, extends C. Experimental, primarily at the University of Illinois[22]. `ftp://a.cs.uiuc.edu/pub/CHARM`.

CLP(*R*). "Constraint Logic Programming (Real domain)." Extends Prolog. In use; available on internet[23, 24]. `http://www.cs.cmu.edu/Web/Groups/AI/html/faqs/ai/constraints/top.html`.

CLOS. "Common LISP Object System." Object-oriented, extends Common LISP[25]. `http://www.cis.ohio-state.edu/hypertext/faq/usenet/lisp-faq/part5/faq.html`.

CLU. Imperative, strongly typed; based on Algol. Pioneered iterators. In occasional use, particularly at MIT[26]. `ftp://ftp.lcs.mit.edu/pub/pclu`.

Concurrent C. Imperative, concurrent; based on C. Includes Ada rendezvous, with guards that can reference formal parameters and sorting expressions. In increasing use[27].

CSP. "Communicating Sequential Processes." Concurrent. Not implemented (but see Occam)[28]. `http://www.comlab.ox.ac.uk/archive/csp.html`.

CST. Concurrent Smalltalk. Object-oriented, concurrent, extends Smalltalk. Experimental[29].

dBASE. Database. Several dialects (dBASE II, dBASE III, dBASE IV) in heavy use[30].

DC++. Concurrent, object-oriented, extends C++. Experimental[31].

DP. "Distributed Processes." Imperative, concurrent. Not implemented[32].

Edison. Imperative, concurrent; based on DP. Experimental[33].

Eiffel. Object-oriented, statically typed, has assertions for axiomatic correctness checking. In use[34, 35]. `http://www.eiffel.com/doc/eiffel.html`. Available from Interactive Software Engineering (ISE).

Eiffel Linda. Object-oriented, concurrent, extends Eiffel and Linda. Experimental[36].

Euclid. Imperative, strongly typed, for systems programming and formal verification; based on Pascal. Several dialects (April Euclid, Small Euclid) in use during the 1980s[37].

Distributed Eiffel. Object-oriented, concurrent, extends Eiffel. Experimental[38].

FORTRAN. "Formula Translator." Imperative, typed, no block structure, weak control structures. Designed at IBM in 1954 under the direction of John Backus. Pioneered arrays, **`for`** loops, and branching **`if`** statements. Various dialects (FORTRAN II, FORTRAN IV, WatFor, WatFive, FORTRAN 66 (ANSI X3.9-1966), FORTRAN 77 (ANSI X3.9-1978), FORTRAN 90 (ISO 1539-1991, ANSI X3.198-1992)) in heavy use since the late 1950s, especially for scientific computing.

http://www.cis.ohio-state.edu/hypertext/faq/usenet/fortran-faq/faq.html.

FP. Functional. Some experimental dialects (FP*, FP*/88N, Berkeley FP) have been implemented[39, 40, 41]. http://www.nectec.or.th/pub/archives/comp.sources.unix/volume20/fpc.

G-2. Multiparadigm, dynamically typed, compiled. Experimental[42].

Gedanken. Clear separation of functional and imperative parts. Not implemented[43].

Logic. Experimental[44]. ftp://ftp.cs.kuleuven.ac.be/pub/logic-prgm/goedel.

Icon. Imperative, strings, backtracking. In use[45, 46]. http://www.cs.arizona.edu/icon/www/index.html.

[incr Tcl]. Scripting, strings, object-oriented, interpreted. Extension of Tcl. In use. http://www.wn.com/biz/itcl.

Intercal. Humorous. Implemented[47]. http://www.nectec.or.th/pub/archives/comp.sources.misc/volume16/intercal.programming language.

Io. Continuations. Not implemented[48].

Leda. Multiparadigm, strongly typed, compiled. Experimental[49]. http://www.cs.orst.edu/~budd/leda.html.

Linda. Concurrent, meant to be embedded in other languages. Embedded in various packages and in use[50]. http://www.cs.yale.edu/HTML/YALE/CS/Linda/linda.html.

LISP. "List Processing Language." Functional, homoiconic. Pioneered garbage collection. In widespread use since the 1960s[51, 52, 53, 54, 55]. There are many dialects of LISP, such as MacLISP, InterLISP, Common LISP, and Scheme. Scheme was designed by Guy Steele and Gerald Sussman. http://www-swiss.ai.mit.edu/scheme-home.html. It has an exceptionally clear and simple semantics and few different ways to form expressions. Common LISP, also developed by Guy Steele, contains a great deal that is not mentioned in Chapter 0, including default parameters, exception handling, a type mechanism, and data structures like strings, arrays, records, and hash tables. http://www.cs.rmit.edu.au/docs/cltl/cltl2.html. Like Scheme, Common LISP uses static, not dynamic, scope rules. The form for defining a function is (**defun** name (param list) (body)). Lambda forms should be quoted: '(**lambda** (x) (+ x 1)); they are invoked by the **funcall** form. Comments start with ; and continue to the end of the line. The print function outputs its parameter. Static scope rules are like those in ML; a scope looks like (**let** ((var1 val1) ...) (body)); use **let*** for recursive declarations.

Lucid. Functional with iteration. Lucid started as a simple, nonprocedural temporal language; it has developed into a programming paradigm called **intensional programming**[56]. http://www.csl.sri.com/Lucid.html.

Lucinda. Linda-Russell hybrid. Experimental[57].

Lynx. Imperative, concurrent, strongly typed; based on Algol. Experimental[58].

Macsyma. Mathematical, interactive. Heavily used during the 1970s and 1980s; still in use and commercially available. http://www.macsyma.com/[59].

Madcap. Experimental. A descendent, Modcap, is in use at New Mexico State University[60].

Maple. Mathematical, interactive. Widely used; commercially available[61].
http://www.maplesoft.com/Maple/.

Mathematica. Mathematical, interactive. Widely used. Commercially available[62, 63].
http://www.wri.com/.

Mesa. Imperative, strongly typed, concurrent; based on Pascal. Used heavily at Xerox Palo Alto Research Center during the 1970s and 1980s[64].

Metafont. Font specification. Widely used[65]. http://etna.mcs.kent.edu/TeX/TeX-FAQ. Part of almost every TeX distribution.

Miranda. Functional, polymorphic types, lazy evaluation. Experimental, in increasing use; commercially available[66, 67, 68]. http://www.cs.nott.ac.uk/Department/Staff/mpj/faq.html#Miranda(TM).

ML. "MetaLanguage." Functional, type inference with polymorphic types, interactive; based on Edinburgh Logic for Computable Functions (LCF). Pioneered type inference. Experimental, in increasing use. ML has evolved into Standard ML[69, 70, 71, 72, 73]. ftp://pop.cs.cmu.edu/usr/rowan/sml-archive/faq.txt. New Jersey Standard ML is interactive, expecting the user to type in expressions, just as shown in Chapter 0. Each expression is terminated by ; . Comments are surrounded by (* and *) . Some useful predefined functions are: use = **fn** : (string **list**) -> unit print = 'a -> 'a Use allows you to read in a program from a list of files. Print allows you to output values. Unit is a type with one value, used as void in C is used. The unary negation operator is ~ .

Modula. Imperative, concurrent, compiled, strong typing; based on Pascal. No longer used[74].

Modula-2. Imperative, concurrent, compiled, strong typing; based on Modula and Pascal. In widespread use[75]. http://www.cis.ohio-state.edu/hypertext/faq/usenet/Modula-2-faq/faq.html.

Modula-3. Imperative, concurrent, compiled, strong typing with structural equivalence, objects; based on Modula-2. Experimental, in increasing use[76]. http://www.research.digital.com/SRC/modula-3/html/home.html.

Oberon. Imperative, strong typing, for students; based on Modula-2. In increasing use[77].
http://www.cis.ohio-state.edu/hypertext/faq/usenet/Oberon-Lang-FAQ/faq.html; also,
http://huxley.inf.ethz.ch/~marais/Spirit.html.

Occam. Concurrent, extension of CSP. In use[78]. http://www.comlab.ox.ac.uk/archive/occam.html.

OPS5. Rule-based. In use[79]. http://www.nectec.or.th/pub/archives/comp.sources.unix/volume12/ops5.

Pascal. Imperative, typed, block-structured; based on Algol-60. In heavy use since the 1970s. Has an ANSI standard[80, 81]. http://www.yahoo.com/Computers/Languages/Pascal.

Perl. "Practical Extraction and Report Language." Scripting, strings, interpreted. In use[82, 83].
http://www.cis.ufl.edu/perl.

Post. Dataflow. Not fully implemented[84].

Prolog. Declarative, logic, patterns, backtrack. In widespread use[85]. `http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/Groups/AI/html/faqs/lang/prolog/top.html`. SICStus Prolog 2.1 is a portable implementation of Prolog; inquiries can be addressed to `sicstus-request@sics.se`. SWI-Prolog comes from the University of Amsterdam. SWI-Prolog is interactive. It begins in query mode, showing a prompt `?-` . To switch to a mode in which facts can be entered, give the query `[user]`. To return to query mode, type an end-of-file. To read facts from a file, give the query `[filename]`. The query `trace` causes prolog to show the rules it tries as the evaluator solves queries. The unary predicate `print` outputs its parameter. The comment delimiters are `/*` and `*/` . To get a bag of all solutions to a query, try `bagof`((list of output variables), query, bagname).

Russell. Types as first-class values. Experimental[86, 87]. `ftp://arisia.xerox.com/pub/russell/russell.tar.Z` .

SAIL. Imperative with some AI structures; based on Algol-W and Leap (a language with associative store). Heavily used at Stanford in the 1970s.

SAL. Imperative, systems administration, database. In use, primarily at the University of Kentucky[88].

Sed. A stream editor standard with all Unix implementations.

Simula. Imperative, types, classes, coroutines; based on Algol. Pioneered abstract data types and object orientation. Various dialects (starting with Simula 67) in heavy use in the 1970s[89].
`http://remarque.berkeley.edu/~muir/free-compilers/TOOL/Simula67-1.html`.

Sisal. “Streams and Iteration in a Single-Assignment Language.” Dataflow; based on Val. in use[90].
`http://www.llnl.gov/sisal/`.

Smalltalk. Object-oriented. Various dialects (mainly of Smalltalk-80) are in use[91, 92, 93].
`http://st-www.cs.uiuc.edu/other_st.html`. A version of Smalltalk 1.0 is available in the Gnu software suite. It is interactive, expecting the user to type in expressions as if they were the body of an anonymous method. The body is terminated by `!` . Comments are surrounded by double quotes. Some useful predefined classes and methods: (FileStream **`open:`** ’file name’ **`mode:`** ’r’) fileIn !1
"read and execute a program from a file"2 anObject **`class inspect`** !3
"show class, superclass, subclasses, methods,4
variables"5 anObject **`printNl`** !6
"print the object with a trailing newline"7 Version 3 of Little Smalltalk is a portable implementation intended for a wide range of machines. It is in the public domain and can be distributed; it is available in `msdos/misclang/stv3-dos.zip` from many sites. Details are available from Tim Budd, Department of Computer Science, Oregon State University, Corvallis, OR 97331. Smalltalk-80 Version 2 is available from ParcPlace Systems, 999 E. Arques Avenue, Sunnyvale, CA 94086-4593, which markets implementations for a wide variety of machines.

SNOBOL. “StriNg Oriented symbOLic Language.” Strings, patterns, dynamic typing, dynamic scope. Pioneered pattern matching. Various dialects (mainly SNOBOL4 and Spitbol) in widespread use in the 1970s[94]. `ftp://cs.arizona.edu/snobol4`.

Specint. Logic, goal-directed. Experimental[95].

SR. Imperative, concurrent; based on Algol. Experimental, in increasing use[96]. `ftp://cs.arizona.edu/sr/sr.tar.Z` .

SQL. “Structured Query Language.” Relational database. Has an ANSI standard (X3.135-1992). In widespread use. `http://waltz.ncsl.nist.gov/~len/sql_info.html`.

Tcl. Scripting, strings, interpreted. In use[97]. `http://www.x.co.uk/of_interest/tcl/Tcl.html`.

Val. Dataflow. Obsolete[98, 99].

References

1. Steven Pemberton, "A short introduction to the ABC language," *ACM SIGPLAN Notices* **26**(2) pp. 11-16 (February 1991).

2. United States Department of Defense, "Reference Manual for the Ada Programming Language," *ANSI/NIL-STD 1815A-1983,* (1983).

3. Raphael A. Finkel, *Constructing and debugging manipulator programs*, Stanford AI Laboratory memo AIM-284, Stanford Computer Science Department report STAN-CS-76-567 (August 1976).

4. J. Hernández, P. de Miguel, M. Barrena, J. M. Martínez, A. Polo, and M. Nieto, "ALBA, a parallel language based on actors," *ACM SIGPLAN Notices* **28**(4) pp. 11-20 (April 1993).

5. Fred Mellender, "An integration of logic and object-oriented programming," *ACM SIGPLAN Notices* **23**(10) pp. 181-185 (October 1988).

6. C. A. R. Hoare, "Hints on Programming Language Design," Stanford CS Department Technical Report STAN-CS-73-403 (December 1973).

7. Peter Naur, "Revised report on the algorithmic language Algol 60," *CACM* **6**(1) pp. 1-17 (1963).

8. A. van Wijngaarden, B. J. Mailloux, J. L. Peck, C. H. A. Koster, M. Sintzoff, C. H. Lindsey, L. G. L. T. Meertens, and R. G. Fisker, "Revised report on the algorithmic language ALGOL 68," *Acta Informatica* **5**(1-3) pp. 1-236 (1975).

9. W. A. Wulf, R. L. London, and Mary Shaw, "Abstraction and verification in Alphard: Defining and specifying iteration and generators," *CACM* **20**(8) pp. 553-563 (August 1977).

10. Luca Cardelli, "Amber," in *Combinators and Functional Programming Languages*, ed. G. Cousineau, P. L. Courien, and B. Robinet, Springer-Verlag, New York (1986).

11. Kenneth E. Iverson, *A Programming Language,* John Wiley and Sons, New York (1962).

12. Kenneth E. Iverson, "A Dictionary of APL," *(ACM) APL Quote Quad* **18**(1)(September 1987).

13. IBM Corporation, *APL2 Programming: Language Reference*, SH20-9227 1987.

14. Barbara Liskov and R. Scheifler, "Guardians and actions: Linguistic support for robust, distributed programs," *ACM Transactions on Programming Languages and Systems* **5**(3) pp. 381-404 (July 1983).

15. Barbara Liskov, "Preliminary Argus reference manual," Programming Methodology Group Memo 39 (et. al.), MIT Laboratory for Computer Science, Cambridge, MA (October 1983).

16. Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *AWK: A Pattern Scanning and Processing Language,* Bell Laboratories, Murray Hill, NJ (September 1978). Gov't. ordering no. et. al. 2nd edition

17. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language,* Prentice-Hall, Englewood Cliffs, NJ (1988). Gov't. ordering no. et. al. 2nd edition

18. Samuel P. Harbison and Guy L. Steele, *C: A Reference Manual*, Prentice-Hall, Englewood Cliffs, NJ (1987). Gov't. ordering no. et. al. 2nd edition

19. Margaret Ellis and Bjarne Stroustrup, *The Annotated C++ Reference Manual,* Addison-Wesley, Reading, MA (1990). Gov't. ordering no. et. al.

20. Scott Meyers, *Effective C++,* Addison-Wesley, Reading, MA (1992). Gov't. ordering no. et. al.

21. George Hockney, Paul Mackenzie, and Mark Fischler, "Canopy Version 5.0," Fermi National Accelerator Laboratory (et. al.) (February 1992).

22. L. V. KALÉ, "The Chare-Kernel parallel programming language and system," *Proceedings of the International Conference on Parallel Processing,* pp. 17-25 (August 1990).

23. N. HEINTZE, J. JAFFAR, AND S. MICHAYLOV, *The CLP(R) Programmer's Manual, Version 1.2,* IBM Thomas J. Watson Research Center (1992). Gov't. ordering no. et. al.

24. T. FRUHWIRTH, A. HEROLD, AND V. KUCHENHOFF, *Constraint Logic Programming — An informal introduction,* Springer-Verlag, Berlin (1992). Gov't. ordering no. et. al.

25. DANIEL G. BOBROW, LINDA G. DEMICHIEL, RICHARD P. GABRIEL, SONYA E. KEENE, GREGOR KICZALES, AND DAVID A. MOON, "Common LISP object system specification," *ACM SIGPLAN Notices* **23**(9)(September 1988).

26. BARBARA LISKOV, R. ATKINSON, T. BLOOM, E. MOSS, J. C. SCHAFFERT, R. SCHEIFLER, AND A. SNYDER, *CLU Reference Manual,* Springer-Verlag, Berlin (1981). Gov't. ordering no. et. al.

27. NAHRAIN H. GEHANI AND W. D. ROOME, *The Concurrent C programming language,* Silicon Press, Summit, NJ (1989). Gov't. ordering no. et. al.

28. C. A. R. HOARE, "Communicating Sequential Processes," *CACM* **21**(8) pp. 666-677 (August 1978).

29. W. J. DALLY AND A. A. CHIEN, "Object-oriented concurrent programming in CST," *Proceedings of the ACM SIGPLAN Workshop on Object-Based Concurrent Programming: ACM SIGPLAN Notices* **24**(4) pp. 28-31 (April 1989).

30. ALAN SIMPSON, *dBASE III Programmer's Reference Guide,* Sybex, Inc., Alameda, CA (1987). Gov't. ordering no. et. al.

31. HAROLD CARR, ROBERT KESSLER, AND MARK SWANSON, "Distributed C++," *ACM SIGPLAN Notices* **28**(1) p. 81 (January 1993).

32. PER BRINCH HANSEN, "Distributed processes: A concurrent programming concept," *CACM* **21**(11) pp. 934-941 (November 1978).

33. PER BRINCH HANSEN, "Edison: A Multiprocessor Language," Technical Report (et. al.), University of Southern California Computer Science Department (September 1980).

34. BERTRAND MEYER, *Object-Oriented Software Construction,* Prentice Hall, Englewood Cliffs, NJ (1988). Gov't. ordering no. et. al.

35. BERTRAND MEYER, *Eiffel: The language,* Prentice Hall, Englewood Cliffs, NJ (1992). Gov't. ordering no. et. al.

36. ROBERT JELLINGHAUS, "Eiffel Linda: An object-oriented Linda Dialect," *ACM SIGPLAN Notices* **25**(12) pp. 70-84 (December 1990).

37. BUTLER W. LAMPSON, JIM J. HORNING, R. L. LONDON, JAMES G. MITCHELL, AND GARY J. POPEK, "Report on the programming language Euclid," *Sigplan Notices* **12**(2) pp. 1-79 (February 1977).

38. L. GUNASEELAN AND RICHARD J. LEBLANC, JR., "Distributed Eiffel: A language for programming multi-granular distributed objects on the Clouds operating system," *Proceedings IEEE 1992 International Conference on Computer Languages*, (April 1992).

39. JOHN W. BACKUS, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs," *Communications of the ACM* **21** pp. 613-641 (1978).

40. A. RADENSKY, "Lazy evaluation and nondeterminism make Backus' FP-systems more practical," *ACM SIGPLAN Notices* **22**(4) pp. 33-40 (April 1987).

41. SCOTT BADEN, "Berkeley FP User's Manual," pp. 2.359-2.391 in *Ultrix-32 Supplementary Documents, Volume II,* Digital Equipment Corporation, Merrimack, NH (1984). Revised

42. JOHN PLACER, "Integrating destructive assignment and lazy evaluation in the multiparadigm language G-2," *ACM SIGPLAN Notices* **27**(2) pp. 65-74 (February 1992).

43. JOHN C. REYNOLDS, "Gedanken," *CACM* **13**(5) pp. 308-319 (May 1970).

44. P. M. HILL AND J. W. LLOYD, *The Programming Language,* MIT Press, Cambridge, MA (1994). Gov't. ordering no. et. al.

45. RALPH E. GRISWOLD AND DAVID R. HANSON, "An alternative to the use of patterns in string processing," *ACM TOPLAS* **2**(2) pp. 153-172 (April 1980).

46. RALPH E. GRISWOLD AND M. T. GRISWOLD, *The Icon Programming Language,* Prentice-Hall, Englewood Cliffs, NJ (1990). Gov't. ordering no. et. al. 2nd edition

47. DONALD R. WOODS AND JAMES M. LYON, *The Intercal Programming Language Reference Manual*, Not published 1973.

48. RAPHAEL L. LEVIEN, "Io: A new programming notation," *ACM SIGPLAN Notices* **24**(12) pp. 24-31 (December 1989).

49. TIMOTHY A. BUDD, *Multiparadigm Programming in Leda,* Addison-Wesley, Reading, MA (1995). Gov't. ordering no. et. al.

50. DAVID GELERNTER, "Generative communication in Linda," *ACM TOPLAS* **7**(1) pp. 80-112 (January 1985).

51. JOHN MCCARTHY, "Lisp 1.5 Programmer's Manual," (et al), MIT Press, Cambridge, MA (1962).

52. SHEILA HUGHES, *Lisp,* Pitman Publishing Limited, London (1986). Gov't. ordering no. et al

53. WARREN TEITELMAN AND LARRY MASINTER, "The Interlisp programming environment," *IEEE Computer* **14**(4) pp. 25-33 (1981).

54. GUY L. STEELE JR., *Common Lisp — The Language,* Digital Press, Bedford, MA (1990). Gov't. ordering no. et al 2nd edition

55. H. ABELSON AND GERALD J. SUSSMAN, *Structure and Interpretation of Computer Programs,* MIT Press, Cambridge, MA (1985). Gov't. ordering no. et al

56. E. A. ASHCROFT AND W. W. WADGE, "Lucid, a nonprocedural language with iteration," *CACM* **20**(7) pp. 519-526 (July 1977).

57. PAUL BUTCHER AND HUSSEIN ZEDAN, "Lucinda — An overview," *ACM SIGPLAN Notices* **26**(8) pp. 90-100 (August 1991).

58. MICHAEL L. SCOTT AND RAPHAEL A. FINKEL, "LYNX: A dynamic distributed programming language," *1984 International Conference on Parallel Processing,* (August 1984).

59. HELLER, *MACSYMA for Statisticians,* John Wiley and Sons, New York (1991). Gov't. ordering no. et al

60. MARK B. WELLS, "Recent improvements in Madcap," *CACM* **6**(11) pp. 674-678 (June 1963).

61. CHAR, GEDDES, GONNET, LEONG, MONOGAN, AND WATT, *Maple V Language Reference Manual,* Springer-Verlag, New York (1993). Gov't. ordering no. et al

62. GAYLORD, KAMIN, AND WELLIN, *Introduction to Programming with Mathematica,* Springer-Verlag, New York (1993). Gov't. ordering no. et al

63. SHAW AND TIGG, *Applied Mathematica, Getting Started, Getting It Done,* Addison-Wesley, Reading, MA (1994). Gov't. ordering no. et al

64. BUTLER W. LAMPSON AND D. D. REDELL, "Experience with processes and monitors in Mesa," *CACM* **23**(2) pp. 105-117 (February 1980).

65. DONALD E. KNUTH AND DUANE R. BIBBY, *The METAFONTbook,* American Mathematical Society (1986). Gov't. ordering no. et al volume C of *Computers and Typesetting*

66. DAVID A. TURNER, "Miranda: A non-strict functional language with polymorphic types," *Proceedings IFIP International Conference on Functional Programming Languages and Computer Architecture*, (September 1985). In *Springer Lecture Notes in Computer Science*, vol. 201

67. DAVID A. TURNER, “An Overview of Miranda,” *ACM SIGPLAN Notices* **21**(12) pp. 158-166 (December 1986).

68. S. J. THOMPSON, “Laws in Miranda,” *Proceedings of the 4th ACM International Conference on LISP and Functional Programming,* (August 1986).

69. ROBERT HARPER, ROBIN MILNER, AND MADS TOFTE, *The Definition of Standard ML,* MIT Press, Cambridge, MA (1989). Gov’t. ordering no. et al Version 2

70. CHRIS READE, *Elements of Functional Programming,* Addison-Wesley, Reading, MA (1989). Gov’t. ordering no. et al

71. ANDREW W. APPEL AND DAVID B. MACQUEEN, “Standard ML of New Jersey,” *Third International Symposium on Programming Language Implementation and Logic Programming,* (August 1991).

72. LAWRENCE C. PAULSON, *ML for the Working Programmer,* Cambridge University Press, Cambridge (1992). Gov’t. ordering no. et al

73. NORMAN RAMSEY, “Concurrent Programming in ML,” CS-TR-262-90 (et al), Princeton University, Department of Computer Science (1990).

74. NIKLAUS WIRTH, “Modula: A language for modular multiprogramming,” *Software Practice and Experience* **7**(1) pp. 3-35 (1977).

75. NIKLAUS WIRTH, *Programming in Modula-2,* Springer-Verlag, New York (1985). Gov’t. ordering no. et al 3rd corrected edition

76. GREG NELSON, *Systems Programming with Modula-3,* Prentice Hall, Englewood Cliffs, NJ (1991). Gov’t. ordering no. et al

77. MARTIN REISER AND NIKLAUS WIRTH, *Programming in Oberon — Steps Beyond Pascal and Modula,* Addison-Wesley, Reading, MA (1992). Gov’t. ordering no. et al

78. D. MAY, “OCCAM,” *ACM SIGPLAN Notices* **18**(4) pp. 69-79 (April 1983). Relevant correspondence appears in volume 19, number 2 and volume 18, number 11

79. L. BROWNSTON, R. FARRELL, F. KANT, AND N. MARTIN, *Programming Expert Systems in OPS5,* Addison-Wesley, Reading, MA (1986). Gov’t. ordering no. et al

80. K. JENSEN AND NIKLAUS WIRTH, “Pascal: User manual and report,” *Lecture Notes in Computer Science,* (18)Springer-Verlag, (1974).

81. ANSI, *American National Standard Pascal Computer Programming Language,* American National Standards Institute, New York (1983). Gov’t. ordering no. et al ANSI/IEEE 770X3.97

82. LARRY WALL AND RANDAL I. SCHWARTZ, *Programming Perl,* O’Reilly and Associates, Sebastopol, CA (1991). Gov’t. ordering no. et al

83. RANDAL L. SCHWARTZ, *Learning Perl,* O’Reilly & Associates, Inc., Sebastopol, CA (August 1994). Gov’t. ordering no. et al

84. CHINYA V. RAVISHANKAR AND RAPHAEL FINKEL, “Linguistic Support for Dataflow,” Computer Sciences Technical Report 136-89 (et al), University of Kentucky−Lexington (January 1989). Also The University of Michigan Electrical Engineering and Computer Science Technical report CSE-TR-14-89, February 1989

85. W. F. CLOCKSIN AND C. S. MELLISH, *Programming in PROLOG,* Springer-Verlag, New York (1981). Gov’t. ordering no. et al

86. ALAN DEMERS AND J. DONAHUE, “Revised Report on Russell,” Cornell CS Department Technical Report TR 79-389 (et al) (1979).

87. HANS-JUERGAN BOEHM AND ALAN DEMERS, “Implementing Russell,” *ACM SIGPLAN Notices* **21**(7)(July 1986).

88. BRIAN STURGILL AND RAPHAEL FINKEL, “System Administration Tools: The SAT Package,” Computer Sciences Technical Report 147-89 (et al), University of Kentucky−Lexington (July 1989).

89. G. M. BIRTWISTLE, OLE-JOHAN DAHL, B. MYHRHAUG, AND KRISTEN NYGAARD, *Simula Begin*, Auerback Press, Philadelphia (1973). Gov't. ordering no. et al

90. JAMES R. MCGRAW AND STEPHEN SKEDZIELEWSKI, "SISAL: Streams and Iteration in a Single-Assignment Language," Lawrence Livermore National Laboratories, Report M-146 (et al) (July 1983).

91. DANIEL H. INGALLS, "The Smalltalk-76 programming system design and implementation," *Fifth Annual ACM Symposium on Principles of Programming Languages*, (1978).

92. ADELE GOLDBERG AND DAVID ROBSON, *Smalltalk-80, The Language and Its Implementation,* Addison-Wesley, Reading, MA (1983). Gov't. ordering no. et al

93. WILF R. LALONDE AND JOHN R. PUGH, *Inside Smalltalk*, Prentice-Hall, Englewood Cliffs, NJ (1990). Gov't. ordering no. et al

94. RALPH E. GRISWOLD, J. POAGE, AND I. POLONSKY, *The Snobol4 Programming Language,* Prentice-Hall, Englewood Cliffs, NJ (1971). Gov't. ordering no. et al 2nd edition

95. JARED L. DARLINGTON, "Search direction by goal failure," *ACM TOPLAS* **12**(2) pp. 224-252 (April 1990).

96. GREGORY R. ANDREWS, RONALD A. OLSSON, MICHAEL COFFIN, IRVING ELSHOFF, K. NILSEN, TITUS PURDIN, AND G. TOWNSEND, "An overview of the SR language and implementation," *ACM Transactions on Programming Languages and Systems* **10**(1) pp. 51-86 (January 1988).

97. JOHN OUSTERHOUT, *Tcl and the Tk Toolkit,* Addison-Wesley, Reading, MA (1994). Gov't. ordering no. et al

98. JAMES R. MCGRAW, "The VAL language: Description and analysis," *ACM TOPLAS* **4**(1)(January 1982).

99. NAHRAIN GEHANI AND C. WETHERELL, *Denotational Semantics for the Dataflow Language VAL,* Internal memo, Bell Laboratories, Murray Hill, NJ (July 1980). Gov't. ordering no. et al