Scientific Programming 15 (2007) 3–8
IOS Press

Scientific programming in Fortran[1]

W. Van Snyder*
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA

Abstract. The Fortran programming language was designed by John Backus and his colleagues at IBM to reduce the cost of programming scientific applications. IBM delivered the first compiler for its model 704 in 1957. IBM's competitors soon offered incompatible versions. ANSI (ASA at the time) developed a standard, largely based on IBM's Fortran IV in 1966. Revisions of the standard were produced in 1977, 1990, 1995 and 2003. Development of a revision, scheduled for 2008, is under way. Unlike most other programming languages, Fortran is periodically revised to keep pace with developments in language and processor design, while revisions largely preserve compatibility with previous versions. Throughout, the focus on scientific programming, and especially on efficient generated programs, has been maintained.

1. Early years

From the time of Babbage until the time of Backus, computer programs were written in machine language, eventually with assistance from assemblers.[2] In that era, computers and their use were expensive, and their capacities were so small that large programs couldn't be written. Squeezing every ounce of performance from the limited resource was worth the labor investment. As computers became larger, more ambitious programs became possible, and the fraction of the cost of computing attributable to labor increased. Realizing this, John Backus and his colleagues developed a FORMula TRANslator to reduce labor costs, and called it FORTRAN. Even though labor's contribution to computing costs was increasing, it was still important to use computers efficiently, so Backus and his team were especially conscious of the need to produce efficient programs. IBM delivered the first FORTRAN compiler for its model 704 computer in 1957 [1]. It was accepted and used by customers because it reduced labor costs without substantially increasing computer usage charges.

IBM eventually offered Fortran compilers for most of their computers, not only the 600 [2], 700 and 7000 [3, 4] series "mainframe" computers, but smaller computers such as 1620 [5] and 1401 [6] as well. IBM's competitors soon offered incompatible compilers. IBM developed revisions, the two most widely used being Fortran II and Fortran IV. Customers wanted to shop around for the best computing value, and manufacturers wanted to lure customers from their competitors. It was time to standardize Fortran.

2. Fortran 66

Nine years after Fortran was introduced, IBM's Fortran IV was a de facto industry base standard. IBM's competitors offered compilers that were mostly compatible with Fortran IV, but tried to differentiate their offerings by providing extensions: Univac [7] and Atlas [8] had Fortran V . . .. The American Standards Association (ASA), the forerunner of the American National Standards Institute (ANSI), chartered the Computer Business Equipment Manufacturers' Association (CBEMA) to develop a standard for Fortran. CBEMA formed committee X3J3, which in 1966 published standard X3.9-1966 [9], commonly called Fortran 66. X3J3 largely standardized the intersection of existing

*Author's current address: Jet Propulsion Laboratory, 4800 Oak Grove Drive, Mail Stop 183-701, Pasadena, CA 91109-8099, USA. E-mail: van.snyder@jpl.nasa.gov.

[1]The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

[2]Ada Augusta Byron, Countess of Lovelace, was Babbage's programmer. She never got a chance to run her programs. Had she had that chance, she might have found a few bugs that historians later found.

ISSN 1058-9244/07/$17.00

practice: There was very little if anything in the 1966 standard that was not already in use in essentially every significant manufacturer's Fortran.

The late 1960's and early 1970's were a time of ferment and controversy in programming language development. Edsger Dijkstra [10] wrote "GO TO considered harmful," and Böhm and Jacopini [11] proved that program control graphs could be transformed to equivalent graphs composed of single-entry single-exit control structures. A revision of Algol developed in 1968 was (and still is) regarded as an elegant tour de force that was unfortunately a bit too complicated. Niklaus Wirth and Tony Hoare realized this during its development, which led them to propose a smaller language known as Algol W [12]. This never caught on, but it did lead to Pascal [13], then Modula and Modula-2, then Ada, Eiffel and other Pascal descendants. The general paradigm being developed in that era was called "structured programming." Numerous preprocessors were developed to provide structured programming in Fortran, including Ratfor [14], S-Fortran [15] and SFtran [16]. In Fortran, all the action was in preprocessors. A revision of the 1966 standard was slow in coming. In addition to Pascal, a language descended from B and BCPL, originally developed at AT&T Bell Laboratories to write operating systems for PDP-8 and PDP-11 computers, was catching on. When AT&T allowed Bill Joy to use the source code for their Unix operating system for his student projects, the cat was out of the bag. Joy went on to be a co-founder of Sun Microsystems, which made substantial inroads into university computer science departments, replacing mainframe computers largely programmed in Fortran and Cobol with workstations largely programmed in C.

Even though computer science teaching was shifting toward the use of Pascal and C [17], scientific programming, and especially numerical analysis, continued to use Fortran. Even by the early 1970's substantial collections of reusable components of mathematical software were available in Fortran. "Legacy" was not the only driving factor in perpetuating the use of Fortran for scientific programming. Manufacturers had made substantial investments in Fortran optimizing compilers. The Fortran H compiler for IBM 360 was widely recognized as an especially outstanding example. Notwithstanding the developments in programming languages during this era, a new standard for Fortran was not produced for eleven years.

3. Fortran 77

Eleven years after the first Fortran standard, CBEMA and X3J3, now under the aegis of ANSI (the successor of ASA), produced the 1977 standard [18]. This was a minor revision of Fortran 66. Very little was added to reflect advances in language design. In particular, notwithstanding that derived types, pointers and dynamic storage had been available in several widely-used languages, they were not added. Importantly, Fortran 77 was a superset of Fortran 66, with very few and trivial exceptions. The CHARACTER data type was added. Some extensions to input/output, including especially OPEN statements and internal input/output were added. The IF THEN ELSE ENDIF control structure was added. Unfortunately, essentially nothing new was done for scientific programming. As a consequence, university computer science departments by and large stopped teaching Fortran, and the teaching of Fortran declined in university engineering departments. Even so, most of the enormous volume of mathematical software that was written during the next decade was written using Fortran 77, as evidenced by the online collection at www.netlib.org.

4. Fortran 90

Until 1976, with the exception of a few one-of-a-kind research computers such as the ILLIAC IV, computers by and large were organized in similar ways. Although the CDC-6600 computer had many of the features we take for granted in modern processors, such as pipelines and caches, it and other production computers executed one instruction or a very small number of instructions at one time, and performed one or a very small number of calculations at one time. In 1976 the first Cray-1 supercomputer was delivered to Los Alamos Scientific Laboratory. Although the ILLIAC IV had been a parallel computer, and had been used at NASA Ames Research Center since 1972, no parallel computer had been produced in numbers. Other computer manufacturers, including ICL, Burroughs and Texas Instruments, were interested in producing computers that performed several computations at once. These "vector parallel" designs stimulated interest to provide facilities in Fortran to exploit these architectures directly.

In 1981, Alan Wilson described [19] his vision of array processing facilities for Fortran. After numerous presentations to X3J3 and WG5, much debate, and some fine tuning, it was incorporated into the design

of what was at the time called Fortran 88, but which because of delays was later called Fortran 8x, and ultimately became Fortran 90 [20]. The justifications were that array processing is a central necessity for scientific computing, and the language features Wilson proposed matched the contemporary architecture of supercomputers.

Array facilities introduced in Fortran 90 include

- access to whole arrays using only their names,
- access to array sections using notation of the form *begin: end: stride*, where each part other than the first colon is optional, with obvious defaults,
- access to array sections using vector subscripts,
- the concept of "assumed shape," whereby the extents of the dimensions of dummy argument arrays are taken from their associated actual arguments,
- array operations without explicit loops, and
- intrinsic functions to inquire the bounds, size and shape of arrays.

Array processing was not the only improvement in Fortran 90 that was aimed at scientific computing. Computations such as evaluation of special functions or integrals, or solution of differential equations, require the program to understand the characteristics of the computer's arithmetic if the last ounce of accuracy is to be extracted without expending unproductive effort. Phyllis Fox and her colleagues at AT&T Bell Laboratories developed the PORT library [21] to improve portability of mathematical software. In particular the function R1MACH could provide the overflow and underflow limits, the round-off level, and the radix, but this function had to be written for each processor. Intrinsic functions to provide these and other quantities related to the computer's arithmetic were added in Fortran 90.

Advances in language design, intended to reduce costs and improve reliability, had been tried in other languages, notably Pascal, Modula-2 and Ada [22]. Features that had proven to be useful without compromising efficiency were added to Fortran 90. Notable among these were

- single-entry single-exit control structures in addition to the IF construct added in Fortran 77, especially a CASE construct, and several variations on a DO construct that does not need a label to indicate its boundary,
- derived types, called record types in Pascal, and
- modules.

Explicit procedure interfaces, usually provided by module procedures, have largely eliminated the errors of too many, too few, or type or rank mismatched arguments in procedure references, while assumed-shape dummy arguments made it easier for processors to detect and report out-of-bounds array references. These were the most common kinds of mistakes in Fortran 66 and Fortran 77 programs.

Memory management had also been a problem in Fortran 77. Dynamic memory had been used in other languages, including Pascal, Ada and C. Fortran 90 added pointers, but with more restrictions than in other languages. In particular, arithmetic operations are not permitted on pointers, and variables that do not have the TARGET attribute are not allowed to be the targets of pointers. This improves alias analysis and register allocation. Allocatable variables in Fortran 90 have even more restrictions than pointers, and allow more aggressive optimizations than pointers. Finally, automatic variables have array bounds or character lengths that are not determined until the procedure in which they are declared begins execution.

The most criticized (and perhaps most irrationally criticized) feature of Fortran was the rigid column boundaries within which statements had to be written. This was addressed by the development of free-form source in Fortran 90. To preserve existing software investments, however, fixed-form source was not removed from the standard.

5. Fortran 95

Fortran 95 [23] was intended as a very minor modification of Fortran 90. It was primarily a "maintenance release," incorporating numerous corrections and clarifications. A few loose ends, including initialization of pointers, were tied up, and a few features from High Performance Fortran, notably the FORALL construct, were added.

6. Technical reports 15580 and 15581

Fortran 95 was developed on a strict timetable, using what the committees called "the train model:" any feature that wasn't ready when the "train" (the Fortran 95 standard) "departed from the station" (was published) would be postponed, or developed as a document that ISO designates a Type-2 Technical Report.

There had been an effort to add a block-structured exception-handling mechanism to Fortran 95, similar to the throw-catch mechanism of C++ or the begin-raise-handle-end mechanism of Ada. Ultimately, agreement could not be reached on the design, or even the desirability of the feature. To provide at last some of the features, it was decided to develop ISO/IEC Type-2 Technical Report 15580 [24] to provide access to features of IEEE arithmetic [25], especially the ability to detect and control exceptional events.

As Fortran 90 came into use, it became evident that some restrictions on allocatable variables were not necessary to allow processors to optimize programs. ISO/IEC Type-2 Technical Report 15581 [26] was developed to allow function results, procedure dummy arguments, and components of derived type objects to be allocatable.

One of the conditions put onto the development of these technical reports was that they would be incorporated into the next revision of the Fortran standard, with changes permitted only if difficulties of implementation arose. Thereby, processor implementers could be confident that their investment in developing these facilities would not be invalidated by the next revision.

7. Fortran 2003

Fortran 2003 [27] was intended to be an ambitious revision. In addition to incorporating Technical Reports 15580 and 15581, the work plan included facilities for object-oriented programming, interoperability with C, program-defined input/output of derived-type objects, and dozens of projects of more minor scope. The summary written by John Reid [28] enumerated 66 features.

The most ambitious of these projects was the development of facilities for object-oriented programming. As with facilities added in previous revisions, every attempt was made not to compromise the ability of processors to generate efficient programs. Although at the time of writing this article there is as yet no experience with the object-oriented programming facilities of Fortran, processor developers who participated in the development believe this objective can be achieved. One facility that has significant potential to improve reliability of scientific computing, interval arithmetic, was unfortunately ultimately not added during this revision.

8. Technical report 19767

Although the module system introduced in Fortran 90 offered substantial improvement for developing programs of modest size, there were still shortcomings in its support for developing large programs. During the development of Fortran 2003, a decision was made to address these shortcomings. So as not to impact the development schedule, this was done by way of Type-2 Technical Report 19767 [29]. This report provides that modules can have submodules similar to Ada packages, which can be divided into a package spec, a package body, and private child units. This reduces compilation and certification cascades, facilitates development and maintenance of large modules, offers flexibility in packaging of library modules, and facilitates software publication while preserving trade secrets.

9. Fortran 2008

Although Fortran 2008 is planned to be a minor revision compared to Fortran 2003, it is much more ambitions than Fortran 95: in addition to incorporating technical report 19767, there are 17 required items and 25 optional items on the work plan, and one that will be implemented as a Type-2 Technical Report [30–32]. The major project of this revision is the addition of co-arrays [33] to Fortran. Co-arrays are designed to facilitate programming multiprocessor parallel computers, as opposed to vector parallel computers. Co-array programming promises to be far simpler than using procedure libraries such as PVM or MPI for parallel programming (thereby requiring less labor cost). Early experience with a prototype at Cray suggests it will result in more efficient programs as well. Once again, Fortran is pioneering a new paradigm for scientific computing.

10. Fortran and scientific programming – Past and future

Fortran was originally designed for scientific programming. The aim was to reduce labor costs without significantly degrading program efficiency. IBM's pre-standardization revisions and those of its competitors, and the five standards and three technical reports that have been published, have aimed at and successfully hit the same targets.

Quality mathematical and scientific software, the kind worth keeping, is usually more expensive per

line than software in other disciplines, at least in part because of the arcane nature of the discipline and the training and experience necessary to develop it. Thus, at least equally importantly, the first standard and each of its revisions have, with trivial exceptions, preserved software investments by maintaining compatibility with previous editions.

Of the thousands of programming languages invented, only a dozen or so have been sufficiently widely used to have had international standards, and only four of those have had revisions that attempted to keep up with language technology while preserving software investments by maintaining compatibility with previous editions: Fortran, Cobol, Ada and C. The C standard committee has voted to put itself into "maintenance" mode, so there will be no further revisions of the C standard. Indeed, the most recent attempt to modernize C was so different from C and so incompatible that it was called a new language: C++. Even counting C, Fortran is the only one of those that was designed for scientific computing, and the one that remains the most appropriate tool for scientific computing.

There are many ideas for new features in Fortran that have not yet been implemented. During the development of the work plan for Fortran 2008, fewer than half of the proposals for new facilities were accepted, and there are undoubtedly other worthy ideas (such as interval arithmetic) that were not proposed. The Fortran committees have no intention of putting themselves into "maintenance" mode. The temperament of the Fortran committees and community, at least at present, is that there will be future developments, which will maintain, and probably enhance, the suitability of Fortran for scientific programming.

References

[1] J.W. Backus et al., FORTRAN Automatic Coding System for the IBM 704, October 1956. http://www.bitsavers.org/pdf/ibm/704/704 FortranProgRefMan Oct56.pdf.

[2] FORTRAN Automatic Coding System for the IBM 704, 1960. C29-4047 http://www.bitsavers.org/pdf/ibm/650/29-4047 FORTRAN.pdf.

[3] IBM 7090/7094 Programming Systems: FORTRAN II Programming, 1963. http://www.bitsavers.org/pdf/ibm/7090/C28-6054-5 FORTRANII Apr64.pdf.

[4] IBM 7090/7094 IBSYS Operating System, Version 13, FORTRAN IV Language, 1966.

[5] 1620 FORTRAN (with FORMAT), 1962. C26-5619 http://www.bitsavers.org/pdf/ibm/1620/C26-5619-1 fortran.pdf.

[6] Fortran specifications and operating procedures for the IBM 1401, April 1965. C24-1455 http://www.bitsavers.org/pdf/ibm/14xx/C24-1455-2 1401 fortran.pdf.

[7] Univac 1100 Series: Fortran V: Programmer Reference, 1966. UP-4060.

[8] C.F. Schofield, A manual of the Atlas FORTRAN V language, 1967.

[9] American Standards Association (now ANSI). Programming Language FORTRAN, 1966. X3.9-1966.

[10] E.W. Dijkstra, Letters to the Editor: GO TO statement considered harmful, *Communications of the ACM* **11**(3) (1968), 147–148.

[11] C.Böhm and G. Jacopini, Flow diagrams, Turing machines, and languages with only two formation rules, *Communications of the ACM* **9**(5) (May 1966), 366–371.

[12] N. Wirth and C.A.R. Hoare, A contribution to the development of ALGOL, *Communications of the ACM* **9**(6) (1966), 413–432.

[13] K. Jensen and N. Wirth, Pascal User Manual and Report. Springer Verlag, 1974.

[14] B.W. Kernighan, Ratfor: A preprocessor for a rational Fortran, *Software: Practice & Experience* **5** (October 197), 395–406.

[15] G. deBalbine, S-Fortran. Technical report, Caine, Farber and Gordon, inc.

[16] C.L. Lawson, The SFtran3 preprocessor, Level 16-d. Technical Report CM-516, Jet Propulsion Laboratory, Section 366, August 1986.

[17] B.W. Kernighan, *The C Programming Language*, Prentice-Hall, second edition, 1988.

[18] American National Standards Institute. Programming Language FORTRAN, 1978. X3.9-1978.

[19] A. Wilson, Array processing – Principles and practice. In ACM 81: Proceedings of the ACM'81 conference, New York, NY, USA, 1981. ACM Press, 33–38.

[20] Organization for International Standardization. Programming Language Fortran, 1991. ISO/IEC 1539-1:1991.

[21] P.A. Fox, A.D. Hall and N.L. Schryer, The PORT mathematical subroutine library, *ACM Transactions on Mathematical Software* **4**(2) (June 1978), 104–126.

[22] Organization for International Standardization. Ada 95 Language Reference Manual, 1995. ISO/IEC 8652:1995(E).

[23] Organization for International Standardization. Programming Language Fortran, 1997. ISO/IEC 1539-1:1997(E).

[24] Organization for International Standardization. Information technology – Programming Languages – Fortran – Floating-point exception handling, 1998. ISO/IEC TR 15580:1998(E).

[25] International Electrotechnical Commission. Binary oating-point arithmetic for microprocesor systems, 1989. IEC 60559: 1989 (Originally IEEE 754-1985).

[26] Organization for International Standardization. Information technology – Programming Languages – Fortran – Enhanced data type facilities, 1999. ISO/IEC TR 15581:1999(E).

[27] Organization for International Standardization. Programming Language Fortran, 2004. ISO/IEC 1539-1:2004(E).

[28] John Reid. The new features of Fortran 2003, November 2003. ISO/IEC JTC1/SC22/WG5 N1579, ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.pdf.

[29] Organization for International Standardization. Information technology – Programming Languages – Fortran – Enhanced module facilities, 2004. ISO/IEC TR 19767:2004(E).

[30] ISO/IEC JTC1/SC22/WG5. Repository of Requirements, January 2006. ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1649.txt.

[31] ISO/IEC JTC1/SC22/WG5. Resolutions of the WG5 meeting on February 13 to 17, 2006 in Fairfax, VA, USA, February 2006. ftp://ftp.nag.co.uk/sc22wg5/N1651-N1700/N1653.txt.

[32] ANSI/INCITS/J3. J3 Work Plan, September 2006. http://j3-fortran.org/doc/year/06/06-010r3.pdf.
[33] R.W. Numrich and J. Reid, Co-arrays in the next Fortran standard, 2005. ISO/IEC JTC1/SC22/WG5 N1642, ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1642.pdf.