The Fortran Story Retold

Selected Reprints 1968-2011

Compiled by Loren Meissner, 2016

CONTENTS

John Backus

Betty Holberton

Edsger Dijkstra

Jeanne Martin

John Backus (1924-2007) led the IBM group that created the first, surprisingly effective, Fortran compiler. Betty Holberton (1917-2000) helped extend Fortran programming language design responsibility to other computer manufacturers and users. Edsger Dijkstra (1930-2002) had a strong impact on public understanding of program structure. Jeanne Martin performed a leading role in the inclusion of International concerns during Fortran language development.

Preface

I was introduced to the magic world of digital computers in the fall of 1952, when my employer sent me for a month to the Institute for Numerical Analysis at UCLA, where I learned to use the prehistoric IBM Card Programmed Calculator – and the one-of-a-kind "world's fastest" SWAC digital computer:

> It had 256 words of memory, using Williams [cathode-ray] tubes, with each word being 37 bits. It had only seven basic operations: add, subtract, and multiply (single precision and double precision versions); comparison, data extraction, input, and output. – WIKI

I coded one or two sample programs in SWAC "assembly language," and I was granted small amounts of computer time to debug them, under the watchful eye of my tutor.

A couple of years later I was employed by the manufacturer of CRC-102 computers, magnetic drum machines that had about 10 times the capacity of SWAC but ran at least 100 times slower. My job was to support the sales force by coding "typical" business applications for potential purchasers. Another member of our Sales Support group told us he heard of a project that was under way, to use computer programs written with mathematical symbols. The project he mentioned was probably Fortran. As described in the introductory History chapter of "Fortran 95 Handbook" (excerpted in this collection):

> In 1954 a project was begun under the leadership of John Backus at IBM to develop an "automatic programming" system that would convert programs written in a mathematical notation to machine instructions for the IBM 704 computer.… This project produced the first Fortran compiler, which was delivered to a customer in 1957. It was a great success by any reasonable criterion. The efficiency of the code generated by the compiler surprised even some of its authors. A more important achievement, but one that took longer to realize, was that programmers could express their computations in a much more natural way.

In 1959 I moved to Berkeley where I was employed at the UC Radiation Laboratory, and I was permitted to spend part time working on my graduate-level education. My 1965 PhD thesis, involving "Non-Linear Resonance," was supported by a Fortran program that simulated the resonance with matrix computations. The matrix sizes that I used were limited by storage capacity.

By the early 1960s, many computer vendors had implemented Fortran compilers that included special features not found in the original IBM Fortran. The American National Standards Institute approved formation of a committee to develop a Fortran standard. Martin Greenfield, who was closely involved with the committee from its beginning, recalls (1987; see excerpt) that the first meeting of the committee (later known as ANSI X3J3 and finally as J3) was held in New York City during August 1962. The Committee's principal accomplishment over the next four years was to choose among various implementations and terminology used by compilers from the various vendors. The first ANSI Fortran standard was adopted in 1966, and became known as Fortran 66.

Popular early Fortran textbooks were written by Daniel McCracken (*A Guide to Fortran Programming*, 1961) and by Elliott Organick (*A Fortran Primer*, 1963; *A Fortran IV Primer*, 1966). I co-authored revisions of the Organick textbook in 1979 (*Fortran 77*) and in 1995 (*Fortran 90*).

In March 1968, the Association for Computing Machinery published a "Letter to the Editor" from Edsger Dijkstra, under the title "*Go To* Statement Considered Harmful." The import of the article is that the sequence of *statements* in a human-originated computer program should bear a predictable relationship to the sequence of *actions* actually executed by the computer. Unlike Fortran 66, an ideal programming language should provide "control structures" to permit description of a computation in terms of *groups* of statements – for repetition, selection, etc. These groups came to be known as *structures*.

As we would say nowadays, Dijkstra's 1968 letter "went viral." One response, which occurred to many Fortran users, was to provide a *preprocessor*. A revised language could be defined, with rules similar to those of Fortran 66 but with structures added. A preprocessor (possibly written in Fortran 66) could read a program written in this "improved" language and produce an "equivalent" Fortran 66 program.

I was interested in this concept. In 1975 I developed one such preprocessor which I named "B4tran." Don Reifer compiled a list of 51 Structured Fortran Preprocessors, and Guy de Balbine compiled a mailing list of persons interested in "the future of Fortran as it relates to Structured Programming."

In February 1975 my employer, the Radiation Laboratory at UC Berkeley, approved my creation of FORWORD Fortran Development Newsletter, to be mailed to interested individuals on request. The focus was on Structured Fortran preprocessors in the first few editions, but later expanded to Fortran language development. The newsletter was later sponsored by ACM SIGPLan and was renamed Fortran Forum. In 1999 I received the ACM SIGPLan Distinguished Service Award "for your significant and lasting contributions to the field of programming languages, especially your efforts on the *Fortran Forum*."

I joined the Fortran Standards Committee (ANSI X3J3) in February 1976. By this time, X3J3 was eagerly working to finish revising Fortran 66, to produce Fortran 77. The IF-THEN-ELSE construct is the only Structured Programming feature listed among "the most significant features introduced" in Fortran 77, according to Fortran Handbook. Other "significant features" of Fortran 77 were the CHARACTER data type and many new input/output facilities, such as direct access files and the OPEN statement. Other extensions were deferred to a future revision.

When Fortran 77 was accepted by ANSI for final processing in October 1977, officers of X3J3 who had guided the 1977 revision – including Frank Engel who had served as Chairman since September 1970 – retired from Committee responsibility.

A new slate of officers was appointed by the parent ANSI X3 body to continue work toward an anticipated further revision – optimistically designated as Fortran 8X and finally named Fortran 90. Jeanne Adams was named X3J3 Chairman and I became Secretary.

Michael Metcalf (see Foreword to *Numerical Recipes in Fortran 90*, in this collection) describes the agonizing process that finally led to Fortran 90:

> The meetings of X3J3 were often full of drama. Most *compiler vendors* were represented as a matter of course but, for many, their main objective appeared to be to maintain the status quo and to ensure that Fortran 90 never saw the light of day. … Most *users*, on the other hand, were hardly

> prepared to invest large amounts of their employers' and their own resources in simply settling for a trivial set of improvements to the existing standard. However, … underlying differences often surfaced … sometimes between users … who wanted Fortran to become a truly modern language and those wanting to maintain indefinite backwards compatibility for their billions of lines of existing code.

A compromise was finally reached at a meeting in Paris during 1988, when ultimate responsibility for Fortran standardization was shifted to the International Standards Organization, with close cooperation between its "Working Group 5" and the American committee X3J3.

> X3J3 and WG5, now collaborating closely, often in grueling six-day meetings, spent the next 18 months and two more periods of (positive) public comment putting the finishing touches to what was now called Fortran 90, and it was finally adopted, after some cliff-hanging votes, for forwarding as a U.S. and International standard on April 11, 1991

In 1998, I wrote a guide to the use of features included in Fortran 90 (and a further minor revision, Fortran 95) for pointers and recursion, especially for sorting and searching applications. A useful feature had been adopted for Fortran in the meantime, at my initiative, extending comparison operator names (for less, greater, equal, etc) from numeric operands to include alphabetic data – so that a subprogram can be written to sort or search data of either form.

Further extensions and revisions to Standard Fortran were published after my retirement. Michael Metcalf states (see 2011 reprint included here):

> Now modern Fortran is a procedural imperative compiled language with a syntax well suited to a direct representation of mathematical formulas. Individual procedures may be compiled separately or grouped into modules. either way allowing the convenient construction of very large programs and procedure libraries. Procedures communicate via global data areas or by argument association. The language now contains features for array processing. abstract data types, dynamic data structures. object oriented programming, and parallel processing.

Fortran may be expected to continue indefinitely as the preferred language for parallel processing of large arrays – whose present size would once have been considered astronomical. Loren Meissner, 2016

Fortran 95 Handbook
MIT Press, 1997 (Pages 1-4):

Jeanne C Adams, Walter S Brainerd, Jeanne T Martin,
Brian T Smith, Jerrold L Wagener

INTRODUCTION

For a programming language, Fortran has been around a long time. It was one of the first widely used "high-level" languages, as well as the first programming language to be standardized. It is still the premier language for scientific and engineering computing applications.

The purpose of this handbook is to describe the latest version of this language, Fortran 95. This chapter sets the stage by providing relevant background and describing the notation used to specify the syntax of Fortran 95.

1.1 History

1.1.1 Initial Development of Fortran

In 1954 a project was begun under the leadership of John Backus at IBM to develop an "automatic programming" system that would convert programs written in a mathematical notation to machine instructions for the IBM 704 computer. Many were skeptical that the project would be successful because, at the time, computer memories were so small and expensive and execution time so valuable that it was believed necessary for the compiled program to be almost as efficient as that produced by a good assembly language programmer.

This project produced the first Fortran compiler, which was delivered to a customer in 1957. It was a great success by any reasonable criterion. The efficiency of the code generated by the compiler surprised even some of its authors. A more important achievement, but one that took longer to realize, was that programmers could express their computations in a much more natural way. This increased productivity and permitted the programmer to write a program that could be maintained and enhanced much more easily than an assembly language program.

About one year after the introduction of the first Fortran compiler, IBM introduced Fortran II. One of the most important changes in Fortran II was the addition of subroutines that could be compiled independently. Thus, Fortran

changed substantially even during its first year; it has been changing continually ever since.

1.1.2 Standardization

By the early 1960s, many computer vendors had implemented a Fortran compiler. They all included special features not found in the original IBM compiler. These features usually were included to meet needs and requests of the users and thus provide an inducement for the customer to buy computer systems from the vendor providing the best compiler. Because the language was very young, a special added feature could be tested to see if it was a good long-term addition to the language. Unfortunately, the profusion of dialects of Fortran prevented programs written for one computer from being transported to a different computer system.

At about this time, the American Standards Association (ASA), later to become the American National Standards Institute (ANSI), began a project of standardizing many aspects of data processing. Someone had the daring idea of standardizing programing languages. A committee, which became X3J3 and is now J3, was formed to develop a standard for Fortran. This standard was adopted in 1966; after the adoption of Fortran 77, it became known as Fortran 66 to distinguish the two versions.

The language continued to develop after 1966, along with general knowledge in the areas of programming, language design, and computer design. Work on a revision of Fortran 66 was completed in 1977 (hence the name Fortran 77) and officially published in 1978. The most significant features introduced this version were the character data type, the IF-THEN-ELSE construct, and many new input/output facilities, such as direct access files and the OPEN statement. Except for the character data type, most of these features had been implemented in many compilers or preprocessors. During this revision, Hollerith data was removed because the character data type is a far superior facility. Although this idea of removing features did not seem very controversial when Fortran 77 was introduced, it proved to be controversial later – so much so that no Fortran 77 features were removed in Fortran 90.

Fortran 77, developed by X3J3, was an ANSI standard – an American National Standard. At about this time the International Standards Organization (ISO) began to mature in the computing language area and adopted Fortran 77

as an international standard; the ISO standard was identical to the ANSI standard, and in fact consisted of one page that referenced the ANSI standard.

As soon as the technical development of Fortran 77 was completed, X3J3 and its ISO counterpart WG5 (SC22/WG5) teamed up for the next revision, which was called Fortran 90. Fortran 90 was an ISO standard first, which the US adopted, word for word, as an ANSI standard. Although X3J3 did the technical work on Fortran 90, and produced the standard document, the torch had been passed as to the "owner" of the Fortran standard; that "owner," for Fortran 90 and forevermore, is ISO.

Fortran 90 was a major advance over Fortran 77. It included: a greatly liberalized source form, a complete set of iteration and selection control structures, enhanced numeric facilities (e.g., the environmental intrinsic functions), a comprehensive data-parallel array language, data structures (including dynamic structures), user-defined types and operators, procedure extensions (e.g., recursion, internal procedures, explicit user-defined generic procedures), module encapsulation (with powerful data hiding features), data-type kind parameters (e.g., to regularize the different "kinds" of reals, provide the corresponding kinds of complex, accommodate different kinds of character, and to resolve overloads in a simple way), dynamic objects (e.g., allocatable arrays), and some I/O extensions (e.g., NAMELIST and non-advancing I/O). The concept of "obsolescent" features was introduced, and a handful of Fortran 77 features were so identified. But removal of significant numbers of archaic features was controversial and so no features were actually removed. A standard-conforming Fortran 77 program is a standard-conforming Fortran 90 program with the same interpretation.

Fortran 95, specified by WG5 and produced by X3J3, represents a minor revision to Fortran 90. Most of the changes correct and clarify what was in Fortran 90. However, a few significant features, such as pure functions and the FORALL construct and statement, were added because they are considered important contributions from High Performance Fortran. A few (but not all) of the features designated as obsolescent in Fortran 90 have been removed from Fortran 95.

1.2 Fortran 95 – The Language of Modern Choice

Fortran 95 should be the language of choice for modern applications development. This section sketches why, and shows how Fortran 95 can serve this role while accommodating the 40-year Fortran tradition (and application base).

Fortran 95 is a minor extension of Fortran 90 – the changes from Fortran 90 are limited primarily to correcting a few errors and inconsistencies in Fortran 90 and strengthening of data-parallel array operations. Fortran 95 therefore supports (most of) Fortran 77 and the vast libraries of such "legacy Fortran code."

Fortran is famous for its efficiency and prowess for numerical computation. These strengths form two of the basic principles guiding Fortran 95. In addition, inherited from Fortran 90, are the principles of high performance data-parallel array operations and efficient data abstraction. These four fundamental principles, the way they are implemented, and their supporting cast of modern features, make Fortran 95 a clean, pleasant language to use for applications requiring either high performance or modern programming techniques, or both.

The Fortran 95 data-parallel array operations constitute a valuable programming paradigm for scalable parallel architectures, especially for array-oriented applications such as many scientific and engineering models.

Dynamically allocatable arrays alleviate many of the deficiencies of those older versions of Fortran with static memory allocation. Pointers provide efficient subobject aliases as well as facilitate the use of dynamic structures such as linked lists and trees. Optimization is preserved in the face of pointers by requiring a target to be designated as such.

Much modern programming involves the definition of arbitrary data types (classes), arbitrary operations on these types, and appropriate data hiding. These capabilities are provided cleanly and efficiently in Fortran 95 by derived data types, flexible procedure overloading and operator definition, and powerful packaging facilities (modules). These provide most object-oriented programming capabilities, except for automatic operator inheritance, in a user-friendly manner. These and more are described in detail in subsequent chapters.

With support of high performance computing, modern programming techniques, and legacy code, Fortran 95 represents an unbeatable combination for application development at the close of the 20th century with its rapidly changing computer technology.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington. D.C. 20234

Date: December 18 1974
To: ANSC X3J3 Members
From: F.E. Holberton
TECH A265
Subject: More on the SAVE Statement

Because I believe the most fundamental fault in the X3.9-1966 Fortran standard is the inability to retain local variables and arrays in subprograms of a standard conforming program, I will mount the soap box once more to express my concern for the future of the Fortran standard.

As one individual who has participated in many areas of the computer field including design of hardware, software and programming languages since its beginning (starting as a programmer on the first electronic computer ENIAC in 1945), I have endeavored to make computers more accessible to a potentially wider user community. To this end, I hope to help make this Fortran standard a language for writing more portable programs and to bring the standard closer to the needs of the user community.

To recite ancient history, in 1954, (twenty years ago) IBM invited interested computer people to gather in many different cities to review and participate in discussions related to the preliminary specifications for the IBM Mathematical FORmula TRANslating System, Fortran, at which time many of us old timers put in our two-cents worth. In 1957, when the compiler (Fortran I) for the IBM 704 was made available for field test, the Applied Mathematics Laboratory at the David Taylor Model Basin (NAVY), where I was then employed, volunteered to be a guinea pig for trial of the system.

With Fortran I for the IBM 704 and A-O (MATHMATIC) for the UNIVAC I in field test at the laboratory, we found no takers among

the professional programmers, because they considered this new approach to “coding” might cost them their future livelihood, so that we were obliged to set up training courses and instigate the first “open shop operations” for the engineers in the other laboratories at this site to try out the systems. This project proved to be a great success and more problems were solved in a shorter time than using mathematicians, who lacked the knowledge of the engineering subject matter, to write programs in assembly code.

The arrival of Fortran II, with the introduction of subprograms, opened up Fortran to a wider class of potential problem solutions. Many programs, particularly the nuclear reactor problems, were too large to be contained in a single memory load of 4096 locations, so that individually tailored overlay techniques were employed so that more storage could be utilized by the data. To make this technique work, it was necessary to map into blank common all of the values that must be saved between the overlay of subprograms, because the local storage for each subprogram resided with that subprogram.

So, for programs that did not exceed a single memory load, all local values were retained, while those programs which were too large resorted to overlay techniques, which were called into action within the program logic.

By 1963, computers were being used in a multi-processing environment and many implementors had a Fortran-like language for use on their systems. While most multi-processing environments saved the last-used-state of a subprogram and data in an automatic overlay system (save and restore), at least one implementor opted to perform automatic overlay of subprograms by loading the initial-state without preserving the last-used-state and therefore all local values were lost. When Fortran came up for standardization in 1963, the specifications were established primarily by the implementors and the concept of “undefinition” was fabricated so that these two worlds could coexist and their products be “standard”. Therefore, today with larger and less expensive memories than in 1957 all users who wish to write portable programs are forced to consider their subprograms as some kind of appendage that can come and go at someone else’s

will; and to be on the safe side he has to consider the worst. Even if the program is a small one with subprograms that can be tucked in a tiny corner of the memory, there is no relief.

If the presence of a SAVE statement in a subprogram just caused the subprogram and its storage to stay around and not disappear, that is one implementation; or it could be automatically saved and returned, that is another solution. To reject the SAVE statement at any level, because of the lack of some sophisticated technique, is totally unacceptable. If users were forced to wait on super techniques, there might never have been a Fortran I. Every year, colleges produce computer science graduates eager for a challenge. When there is a need, some solution is usually found.

It is questionable whether 25 people on a standardization committee can force the rest of the world to change. The market place calls the shots. However, in 1976, ten years after the first Fortran standard, when the SAVE statement is part of the Fortran language, a large segment of the Fortran community will realize for the first time that the “portable” programs that worked on most systems weren’t really “standard”. At this time, hopefully, they won’ t object to inserting the SAVE statement in their programs to rectify a discrepancy that existed in the standard, but not in most -implementations.

Annals of the History of Computing
Vol 6, No 1, Jan 1984
Special Issue: Fortran's Twenty-Fifth Anniversary

Institutionalization of Fortran

(Pages 28-32)

Jeanne Adams, Chair

CONTENTS

Early Fortran User Experience

I want to mention a little-known aspect of how Westinghouse-Bettis, a nuclear-power-reactor development laboratory, together with a lot of other groups who ultimately became known as "the nuclear-codes crowd," got interested in large-scale computing in the middle 1950s.

Systems of elliptic partial differential equations are used to describe fixed-geometry nuclear-power reactors for criticality calculations. One group, reputed to be the world's "outstanding authorities," investigated the use of digital computers to perform such calculations by relaxation or successive-approximation techniques. They concluded on theoretical grounds that the rate of approach to a correct solution, which must decrease with problem size, went to zero for numerical models of order 600. Using what was then the world's most powerful computer, the NORC (Naval Ordnance Research Calculator), they performed experiments that seemed to support that conclusion.

Problems of the order 2500 were already in use for two-dimensional reactor design work, represented by passive electric-network models. Using a special-purpose analog simulator, one such solution took about six weeks of day-and-night chain-gang-style labor for several skilled technicians. The 600-limit "proof" had pretty well convinced the reactor designers that digital computers weren't going to help them.

A team of mathematicians, headed by Elizabeth Cuthill at the U.S. Navy's David Taylor Model Basin near Washington, concluded that the proof applied

to the mathematical technique instead of to the problem. Using a new technique, they wrote a program to solve problems up to Order 2500.

The machine they had available was a UNIVAC I computer that had about as much memory as a modern pocket-size key-driven calculator and executed roughly 1000 instructions per second. The program took between 30 and 40 hours of machine time per solution, but it ran! Results were correct and usable. It was used to design several reactors.

Although Betty would never have named anything after herself, her 2500-point program became known as the Cuthill Code – now a household word in the nuclear-codes community. Without such a successful demonstration that the world's outstanding authorities could be wrong, there would have been no early large-scale nuclear codes. The demand for more and more powerful computers would not have gained a major push.

When I recently discussed this development with a distinguished computer historian, I was startled to learn that few people in other fields of applied mathematics have even heard of the work. As of today, he will no longer be able to make that statement.

Criticality calculations gave information for a particular design and a particular set of operating conditions about the extent to which the chain reaction was supercritical. These calculations took a lot of machine time, and memory space was extremely expensive, so they were hand-polished to maximum efficiency in time and space. In 1957 one- and two-dimensional versions were running on an IBM 704.

To simulate the reactor core through its working lifetime, it was necessary to perform a depletion or burnout calculation using the output of a criticality calculation to determine at each point inside the core what neutron bombardment had done to the core materials. That turned out to be an enormously complex problem. For a 250-point one-dimensional solution that was running at that time, for example, the depletion calculation included 30,000 lines of assembler code. The core designers were planning two- and three-dimensional codes.

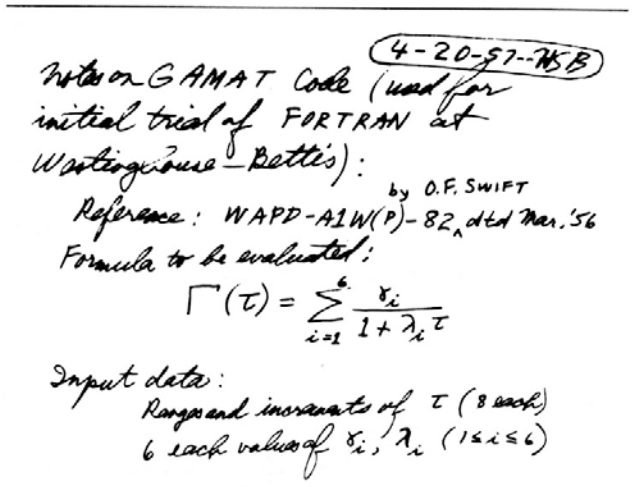
Understanding of the behavior of the materials under nuclear bombardment grew rapidly. This further complicated the coding problem, which, of course, was accompanied by a huge maintenance problem. The question arose: Could we do that much coding and maintenance?

As if that weren't enough pessimism, the people in the Mathematics Department at Bettis Laboratory were pessimistic about the still-fetal FORTRAN. We had expected that Fortran, presuming it would be available some day, could never construct code that was really efficient, either in time or in space. Our intention to take a look at Fortran was accompanied by the assumption that it was going to produce rotten code – as a matter of fact, on occasion it did. Some of the Fortran object code was amazingly efficient, but we hadn't yet learned how to predict or control that aspect of compiling.

Fortunately, the depletion calculations only got executed once per time step. Unlike the criticality calculations, although they took a lot of code, they didn't have to be efficient; they only had to be correct. To our delight, Fortran produced *correct* code, and the amount of labor required to debug and maintain the code – and even to change it substantively – was remarkably small.

In the first issue of the Annals [Volume 1 Number 1 (July 1979), pp. 72-74], I described a Fortran test problem that was part of a depletion calculation. If the solution was correct – even if its resulting code was inefficient – it would be important; this was not just an exercise.

The expression shown for gamma in Figure 1 was computed by incrementing several variables to generate a table for what was known as "gamma of tau for the inhour formula." The independent variable was the amount of time each material was in the reactor core under neutron bombardment. The result was used to calculate the behavior of each point in a geometric array of material as a function of time. The story in the *Annals* gave some operating details of our first test of Fortran using that calculation.


4-20-57--WSB

Notes on GAMAT Code (used for initial trial of FORTRAN at Westinghouse-Bettis):

by O.F. SWIFT

Reference: WAPD-A1W(P)-82, dtd Mar. '56

Formula to be evaluated:

$$\Gamma(\tau) = \sum_{i=1}^{6} \frac{\gamma_i}{1+\lambda_i \tau}$$

Input data:

Range and increments of τ (8 each)

6 each values of γ_i, λ_i $(1 \le i \le 6)$

Figure 1. Notes on Gamma (Tau) code.

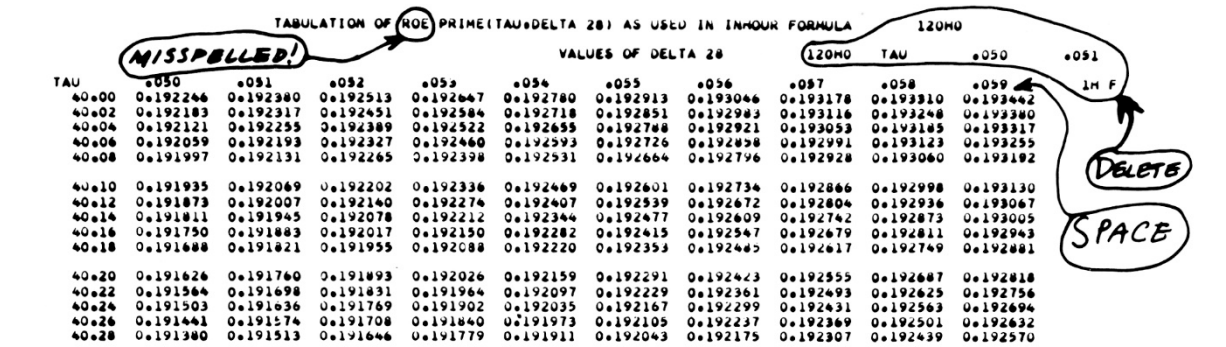
Late one Friday afternoon – the Friday before a SHARE meeting – the Bettis mailman showed up with an unmarked box of cards with no documentation. Lou Ondis, Ollie Swift, and I were standing in a hallway when the mystery

package came along. Unfortunately Jim Callaghan's carpool had already removed him from the scene. Ollie had written a report specifying the "gamma of tau" calculation, on the basis of which Jim had written a Fortran test program. Jim had only this shiny gray thing marked "Fortran Programmer's Manual," a sort of fat brochure that in retrospect was incredibly accurate in comparison with typical modern documentation. Jim had spent about one afternoon writing this program. To give you a comparison with our previous methods, we later estimated it would have taken about two weeks to have written this amount of code in assembly language and another week or two to debug it.

Lou suspected that the anonymous box of cards might be the overdue Fortran compiler. Ollie suggested a way to find out. Hang a full set of 10 blank tapes on the 704 and act as though we believed this was, in fact, Fortran. Load the compiler and the source program – it did not require input data, because one set of test values was built in – and attempt to compile, load, and execute.

Lou got through those processes successfully. After a few minutes of machine activity, we wound up with a single, printed, English-language diagnostic of incredible specificity. Figure 2 gives the "diagnostic program results."

We looked at the card. The diagnostic was right! Lou reproduced the card with a comma stuck in the right place. We recompiled. After a little whiff of computing, there came something like 28 pages of output (see Figure 3).



TAU	.050	.051	.052	.053	.054	.055	.056	.057	.058	.059
40.00	0.192246	0.192380	0.192513	0.192647	0.192780	0.192913	0.193046	0.193178	0.193310	0.193442
40.02	0.192183	0.192317	0.192451	0.192584	0.192718	0.192851	0.192983	0.193116	0.193248	0.193380
40.04	0.192121	0.192255	0.192389	0.192522	0.192655	0.192788	0.192921	0.193053	0.193185	0.193317
40.06	0.192059	0.192193	0.192327	0.192460	0.192593	0.192726	0.192858	0.192991	0.193123	0.193255
40.08	0.191997	0.192131	0.192265	0.192398	0.192531	0.192664	0.192796	0.192928	0.193060	0.193192
40.10	0.191935	0.192069	0.192202	0.192336	0.192469	0.192601	0.192734	0.192866	0.192998	0.193130
40.12	0.191873	0.192007	0.192140	0.192274	0.192407	0.192539	0.192672	0.192804	0.192936	0.193067
40.14	0.191811	0.191945	0.192078	0.192212	0.192344	0.192477	0.192609	0.192742	0.192873	0.193005
40.16	0.191750	0.191883	0.192017	0.192150	0.192282	0.192415	0.192547	0.192679	0.192811	0.192943
40.18	0.191688	0.191821	0.191955	0.192088	0.192220	0.192353	0.192485	0.192617	0.192749	0.192881
40.20	0.191626	0.191760	0.191893	0.192026	0.192159	0.192291	0.192423	0.192555	0.192687	0.192818
40.22	0.191564	0.191698	0.191831	0.191964	0.192097	0.192229	0.192361	0.192493	0.192625	0.192756
40.24	0.191503	0.191636	0.191769	0.191902	0.192035	0.192167	0.192299	0.192431	0.192563	0.192694
40.26	0.191441	0.191574	0.191708	0.191840	0.191973	0.192105	0.192237	0.192369	0.192501	0.192632
40.28	0.191380	0.191513	0.191646	0.191779	0.191911	0.192043	0.192175	0.192307	0.192439	0.192570

Figure 3. Gamma (Tau) output (first part of 28 pages).

There were several errors in our use of Roy Nutt's FORMAT phase, but the results rang like old crystal. We random-spotchecked about 15 values. I was convinced that all of the output was essentially good to the six decimal digits printed. We remarked in the *Annals* (our story had first been published in 1971)

that a couple of hundred compiler fixes down the road, it was hard to believe it had happened. I still feel that way.

John Backus has commented that although his Fortran group intended to distribute the first Fortran compiler in binary-card form, only one or two decks actually got punched. They used up several reproducing card punches per binary deck produced; the machinery couldn't stand the mechanical load. The fact that the newborn Fortran got to us on the last working day before a SHARE meeting – and that Jim had produced a workable test program that was ready to try the compiler – represented incredible, blind good luck.

The Emergence of Fortran IV from Fortran II

My subject is slightly broader than the emergence of 704 Fortran II to 7094 Fortran IV. I'm going to talk about the evolution of 704 Fortran during the period from 1957 to 1964 from my personal viewpoint. During this period I had various responsibilities in connection with Fortran.

My first responsibility was to assist on the transfer of the Fortran project from the Programming Research Group under John Backus to the Applied Programming Department. Later I was manager of 7094 programming, and still later I was responsible for coordinating Fortran processor implementations within IBM. In 1957 the status of Fortran was that the initial compilers were completed by the Programming Research Group, which had embarked on a significant improvement called Fortran II that has enabled users to break up the program – a large application – into independent compilations. This was an important advance to which attention should called. In fact, it was the genesis of many of the linking loaders we have today. The idea of having an application program written not as the output of a single compilation but of many was new. It greatly expanded the possible use of Fortran because it meant that if some small part of the application required assembly-language programming it could be done without writing a separate routine or function in the Fortran language.

When I became involved with the Applied Programming Department, there were approximately 10 people to take over the work of Backus's Programming Research Group. Most of these people were capable but junior in experience in programming. Our first responsibility was to learn the structure of the compiler. Backus's group had an informal management style, and there were some things that bothered us a little. For example, the different sections were

written in two different assembly languages – certain sections in one and certain in the other. When we finally got Section 2, the Programming Research Group had lost the symbolic code so it came over to us in absolute.

The most important initial project undertaken in Applied Programming was to get a version ready for the IBM 709, which had been announced in January 1957 and was first shipped late in 1958. Because the group was new, a minimum number of changes were made in order to make Fortran operative on the 709. This machine had different input/output, and the configuration we chose to support was 8K main memory (instead of 4K) with a drum. The 8K main storage meant we had to be a little bit careful in shoehorning everything into storage.

The original plan for the 709 programming support was to have a SHARE Operating System (SOS) designed by the most experienced users in SHARE. It was basically a design to surround the assembly language program with some nice debugging tools. One group in programming would work on SOS, and the Fortran work would go on in parallel. The initial thinking was that we would integrate Fortran within SOS. We ran into schedule difficulties. There were some technical difficulties, too, in that the Fortran II approach of modular programs was not well matched with the format of the deck of SOS. There was some allowance to match the two, but I don't think all the technical aspects had been worked through. In any case, we weren't able to integrate. The first 709 Fortran came out as a stand-alone system, not within SOS, and used a loader very much like the original linking loader of the Fortran system on the 704.

The 704 and 709 Fortrans were successful quite early – especially Fortran II – but the penetration on users, so to speak, was rather uneven. The most experienced users (who dated from the days of the IBM 701) tended to retain assembly language programming, and the newest and least sophisticated newcomers to computing were most frequently Fortran users. Nonetheless, the technical basis of Fortran was sufficiently sound that usage was like a snowball going downhill.

Soon there were hundreds of customers making hundreds of suggestions for improvements. They would find bugs and send them in – not only error reports, but in many cases the fixes would come in along with the reports. Many suggestions applied to such matters as improvement of diagnostics – little practical

things – and it was as if there were hundreds of people working on improving Fortran. The suggestions just poured in, and we put them in as fast as we could.

A significant event occurred in 1958. The German Applied Mathematics Society (GAMM) proposed to the Association for Computing Machinery (ACM) that an international algorithmic language be developed, and SHARE requested that Backus be its representative. He participated in that effort and gave a report in the fall of 1958. As a result of this report, SHARE was very enthusiastic about the possible future of ALGOL. In fact, SHARE went so far as to pass a resolution requesting IBM to implement ALGOL.

During a period of about a year and a half when we were making minimal improvements on Fortran, we were also working up an ALGOL experimental compiler. After about two years, IBM and SHARE jointly realized that ALGOL was not going to supersede Fortran, and that we should look toward longer range improvements in Fortran. We decided to clean up Fortran II; this was the basis of the transition from Fortran II to Fortran IV. The cleanup consisted of a lot of details such as getting rid of machine dependent irregularities of the language, and introducing and straightening the treatment of COMMON and EQUIVALENCE so that customers didn't have to have special courses on how to write EQUIVALENCE statements. Many changes were planned.

One important limiting factor, of course, was that we wanted customers who had Fortran II programs to be able to preserve them. SHARE planned and wrote a translator written in Fortran to translate from Fortran II to Fortran IV. Don Moore, Jay Allan, and Paul Rogoway wrote that program, [J J Allen, D P Moore, and H P Rogoway, "SHARE Internal Fortran Translator (SIFT)," *Datamation* 9, 3 (March 1963), 43-46] and it was used successfully on the conversion of Fortran II to IV.

Early Fortran at Livermore

The Lawrence Livermore National Laboratory (LLNL), located about 40 miles due east of San Francisco, is a facility for nuclear research and weapons design. Being only slightly younger than the modern digital computer, LLNL's history is closely tied to that of the computer industry in that it is:

1. A leader in the application of computers (and Fortran) to the solution of large-scale scientific problems and to major systems software implementations.
2. Staffed by experts in both software and hardware design.

3. One of the largest concentrations of computing power in the world, housing both the Octopus Computer Network and the Magnetic Fusion and Energy Computer Center. The latter is a national network.

LLNL has a user community of 8000 employees, of whom 4000 are scientists or engineers. It has 2000 time-sharing terminals, and works on scientific applications in mathematical physics and biomedical research. Its system software consists of operating systems, language processors, and computer graphics.

Computing at LLNL began with the first commercially available machines, the UNIVAC I in 1952 and its successors, the IBM CPC [Card-Programmed Calculator], an IBM 701 in 1954, and two IBM 650s. There were some early compiler efforts. Kl and K2 were experimental algebraic compilers for the IBM 701 based on flowchart algorithms. K3 was an IBM 704 compiler designed to maintain the integrity of conventional mathematical notation. It required three cards per statement, the first and third being used for exponents and subscripts. was named K3 for "Kent Ellsworth and the world's third compiler." K3 had a successful first run. It then became the world's second Spruce Goose in the wake of Fortran's growing popularity.

Interest in Fortran began in 1955, when IBM announced plans for an automatic coding system for the IBM 704 (LLNL eventually had four 704s). In those early days, LLNL was one of the few organizations that used computers and was aware of the Fortran project. Sidney Fernbach, head of the Computation Department, spearheaded an effort to gain firsthand knowledge of Fortran's implementation and potential as a programming aid. I was sent to New York in the summer of 1956 to work with the Fortran development team, led by Backus.

From the advent of Fortran in early 1957, an extended Fortran called LRLTRAN became the most used programming language at LLNL. It was typically used to compile compilers (Fortran-Fortran) and to maintain up-to-date software for succeeding generations of LLNL's large mainframes.

The first Fortran-Fortran was that of the IBM 709 (LLNL had two in 1963) – the first LRLTRAN compiler – with no extensions to the language. The first two extensions appeared with the Livermore Automatic Research Computer (LARC) (1960), a decimal machine contemporary with the IBM 7030 (Stretch). The LARC's Fortran compiler came from the new Computer Science Corporation and allowed a parameter statement with symbolic names for declarative

constants, and alphanumeric and numeric statement labels. Early Fortrans lacked mixed mode arithmetic or byte declarations; the latter shortcoming was decried by system programmers who felt “betrayed” by the language designers. IF THEN ELSE was added in 1977. Most of LRLTRAN extensions are now “standard” under ANSI Fortran 77 specifications. Thus, after years of new user comments such as, “That’s not Fortran,” LRLTRAN is again Fortran (well, almost).

Annals of the History of Computing
Vol 6, No 1, Jan 1984
Special Issue: Fortran's Twenty-Fifth Anniversary

Meetings in Retrospect

Fortran Activities at SHARE Meeting

Pages 65-69

Elliott C Nohr

The Fortran exhibit displayed at NCC '82 and the IBM Santa Teresa Laboratory was subsequently shipped to the SHARE 59 meeting in New Orleans, August 22-27, 1982.

At a special Fortran session chaired by John Ehrman, Elliott Nohr of IBM [General Products Div., San Jose] spoke about the early days of Fortran and how SHARE and IBM worked together to make it more widely used.

We are presenting an edited version of Nohr's paper with SHARE's permission.

...

SHARE XIV was held in Los Angeles in February 1960; Donn Parker was chairman of the Fortran committee. The topic being discussed then was whether assembly-language instructions should be permitted in the middle of Fortran programs. It was argued that this would improve the efficiency, and it was also argued that one would lose all compatibility. After a prolonged floor argument, SHARE members agreed that symbolic instructions should not be added. It is true that IBM had something called Fortran III, which was really Fortran II with symbolic instructions added, that had been distributed to a very few people. By this time, Fortran had a large number of active users.

In August 1960, George Mealy (RS-Rand) wrote to the Fortran chairman on the subject of "Whither Fortran," Some quotes from this letter are:

> The committee has proceeded on a course of jacking Fortran up inch by inch; a bit more leverage has been required at each step.

> We are rapidly reaching the point at which only very minor improvements can be made within the existing framework. How do the IBM people feel about spending their lives patching things up rather than being free to do more creative work? In short, I think we should stop trying to kid Fortran into working better and completely rewrite it.

SHARE XV was held in Pittsburgh in September 1960, and the committee met in closed session starting on Sunday afternoon with 26 of the 28 members present. On that day we were discussing such things as the *G*-type format by Jim Porter (General Electric); the debug package by Bill Hefner (General Electric), Fred Scaife (Martin Aircraft), and Tom Martin (Westinghouse Electric); the WD buffered I/O routine; the DE-FAP routine; and the General Electric 709 Fortran format generator by Dorothea Clark. It was suggested that all of these should be distributed through the SHARE secretary for field tests. SHARE decided to appoint a "czar" to oversee the testing of all of the customer-developed extensions.

After SHARE XV, Bruce Rosenblatt was appointed chairman of the group. The Fortran group was invited back to New York in January 1961 to listen to a report by the IBM Fortran planning group. The ideas generated at this meeting resulted in a preliminary specification of Fortran IV.

At SHARE XVI, held in San Francisco in March 1961, the preliminary specifications were submitted and discussed. This was the beginning of 7090 Fortran IV. One of the differences from the old Fortran I/II and Fortran IV was that all of the arrays were stored backward in Fortran I/II. That is, they started with the highest address and worked back toward the beginning, while the programs started with the lowest address and worked toward the end. If the two ever met, you had some problems. The backward arrays also caused problems when working with other subroutines in assembler language; you had to keep remembering that the arrays were actually in reverse order. The new Fortran IV storage was in the forward direction. The EQUIVALENCE statements were not allowed to use multiple subscripts; therefore, you had to figure out which element of the array you wanted. If you wanted a 10 × 10 array, you could not say the fifth element of the third row; you had to indicate it as if it were a scalar variable and count which number it was; this led to many errors. Also, at this time, there were discussions of the label COMMON, adjustable dimensions, full-word integer arithmetic, and logical IFs as part of the new Fortran IV.

In 1961, SHARE urged IBM to release its new language called COBOL 61. To digress for a minute, IBM had a commercial language available called Commercial Translator (COMTRAN) that was one of the languages considered when COBOL was developed, but customers were requesting IBM to provide a COBOL.

Jim Porter (General Electric) had agreed at an earlier date to work on a format generator because the Fortran FORMAT statement only had an *n*H character to put in character data. This was the source of many errors. General Electric worked on a format generator, but IBM decided not to include it in the IBM system,

During 1961, some SHARE members felt that they were having some difficulty with IBM. A proposal was made that:

> The SHARE/Fortran Standards and Evaluation subcommittee wishes to report to the executive board that its ability to communicate with IBM applied programming is rapidly deteriorating. This is essentially true in the area of language modification. Decisions in this area are filtered through a group that placed undue emphasis upon compatibility with systems for non-SHARE machines and is consequently unsympathetic to our needs.

…

At that meeting it was obvious that the new Fortran IV that was being discussed was going to be significantly different from Fortran II and that a program would be needed to convert Fortran II to Fortran IV.

It was decided that a new committee would be formed to take all current practices and convert them to the new Fortran IV. This project was called the SHARE Internal Fortran Translator (SIFT). The members of the committee were Jay Allen of IBM, Don Moore of UCLA, and Paul Rogoway of Aerospace Corporation. The deadline for the conversion project was January 1962, but the project took until September 1962, at which time IBM accepted the conversion program SIFT for maintenance and distribution.

In 1962, Fred Scaife (Martin Aircraft) became the chairman of the Fortran committee. In 1963, the Fortran committee set up an advanced-language planning committee to look at an extended Fortran IV. This committee was known as the 3×3 committee. It was composed of three SHARE members and three IBM members. The SHARE members were Bruce Rosenblatt (Standard Oil),

Hans Berg (Lockheed Aircraft), and Jim Cox (Union Carbide-Oak Ridge). The three IBM members were George Radin, Bernice Weitzenhoffer, and C. W. Medlock. The committee soon realized that in order to make the extensions needed, they could not keep compatibility with Fortran. The language that resulted was PL/1.

By this time, the IBM 360 system had been announced, and SHARE members were concerned about getting their programs to run on the 360; it was a 32-bit machine, and they had a 36-bit machine. In particular, they were concerned with the accuracy of their floating-point numbers and how to handle the Hollerith constants that were then six characters per word and now would be four characters per word. (We also were going from a 6-bit character to an 8-bit character.)

One last item worth mentioning is character variable type. This was introduced in February 1967 at SHARE XXX. As most of you are aware, IBM did listen and implement it, even though we had to wait 14 years.

Library of Association for Computing Machinery

From Proceedings, AFIPS National Computer Conference
Houston, June 1982 (Pages 817-824)

History of Fortran Standardization

Martin N. Greenfield

ABSTRACT

The history of Fortran standardization, ranging from the original efforts in the early 60s up to the present, is presented. Some of the precedent-setting development during the initial cycle in handling problems common to all language standardization is discussed. The background in introducing some of the features in Fortran 77 is covered. The nature and reasoning behind the current activity are described.

There is an interesting and appropriate introduction in my daughter's college text on Fortran. It reads, "After you have learned some of the language, you will show off your sophistication by knocking its lack of elegance. Everybody does. After you learn a little bit more, you will appreciate that it is the way to really get your work done."

Fortran has for most of its life been the blue-collar worker of the programming language set. What it lacked in *savoir-faire* and style, it returned in cost effectiveness. Those working with Fortran pioneered the way for the acceptance of higher-level languages and their standardization. Those who have influenced its development were continually aware of the underlying fact that the language, first and foremost, must remain an efficient tool for producing results.

Fortran standardization dates back to early 1960. The language had just been selected by industry over ALGOL as the language for scientific and engineering work. The major vendors recognized the requirement to provide Fortran compilers in order to compete with IBM. The general strategy was to provide a compiler with the functionality of the 704/709 Fortran and to add features as a competitive inducement. The impact of these added features was two-edged. Although they contributed to the development of the language, they threatened to splinter it into a myriad of uncontrolled dialects. Adding to the problem, a rigorous definition of the language did not exist, even within IBM.

Fortunately, at that time ASA [American Standards Association] (subsequently to become ANSI [American National Standards Institute]) and BEMA [Business Equipment Manufacturers' Association] (subsequently, CBEMA [Computer and Business Equipment Manufacturers' Association]) undertook sponsorship of a massive standardization effort covering a broad variety of data processing areas. Someone had the brave idea of including languages. The ASA X3.4 committee conducted a survey of existing programming languages. Fortran, COBOL, and ALGOL were selected as the candidates for standardization. X3.4 at their May 1962 meeting established the X3.4.3 committee and directed it to standardize the Fortran language.

INITIAL STANDARDIZATION (1962-1966)

Bill Heising, of IBM, was appointed as the initial chairman of X3.4.3. Bill sent invitations to potentially interested groups to attend a formation meeting. Accompanying the invitations was a document written by Bill together with Dick Ridgeway that was proposed as the starting draft for the standardization effort. This Heising-Ridgeway Fortran was based upon the forthcoming [IBM] Fortran IV.

The initial meeting of X3.4.3 was held at the BEMA Headquarters in New York City on August 14, 1962. This makes 1982 both the twenty-fifth anniversary of Fortran [see Annals of the History of Computing, Vol 6 No 1, January 1984: Special Issue – Fortran's Twenty-Fifth Anniversary: Pioneer Day, Houston, 9 Jun 1982] and the twentieth anniversary of the start of its standardization. At this August 1962 meeting, there was a consensus to undertake the standardization work. The scope and criteria of the effort were established.

X3.4.3 assumed the role of parent and policy maker and delegated all the chores below that to two working subcommittees. As such, X3.4.3 met only about twice a year. X3.4.3 originally had about two dozen regular members. All the major hardware vendors were represented. A number of user groups (SHARE, Honeywell Users Association, USE, VIM, IBM 1620 Users, CO-OP) participated. Some software houses (CSC [Computer Sciences Corporation], CUC [Computer Usage Corporation]) and universities (Wisconsin, Penn State, UCSD [University of California, San Diego]) had members.

The parent X3.4.3 did thrash out some very controversial issues. One of recall concerned a proposal from those working with the then new character-addressable hardware. They could save much space by not allocating the same space to integer and logical data as was allocated to reals. In fact, they preferred not to have any fixed storage relationship between the data types. Logicals could be packed into one byte or less. Double precisions could occupy just two or three more bytes than reals. Their arguments centered about the concept that a language standard should not be as hardware biased as the word-storage-unit relationship implies. After some impassioned discussions the heavy dependence of Fortran on storage association for efficiency and the dominance of word addressable processors won.

Most of the actual standardization work was handled by the two subcommittees. X3.4.3-IV was responsible for the standardization of the language based on Fortran IV, while X3.4.3-II was to do the same for Fortran II.

The subcommittees were small compared to the size of groups currently developing draft standards. It was fortunate, because it provided an efficient working arrangement and uninterrupted participation. Little time was lost in having to bring new members up to date. The regular members of X3.4.3-IV were

Martin N. Greenfield, Honeywell, chairman
Richard K. Ridgeway, IBM, editor
Caral Sampson (Giammo), Philco, secretary
Tom Martin, SHARE and Westinghouse
Geraldine Zimmerman (Bowen), UNIVAC
Lou Gatt, CSC
Ken Tiede, CDC
Carl Bailey, CO-OP and Sandia
Bob Mitchell, CO-OP and UCSD

Along with the X3.4.3 chairmanship responsibilities, Bill Heising was a very active participant in the effort of the X3.4.3-IV subcommittee. Others from X3.4.3 participated from time to time, but the bulk of the effort was done by the group above.

The work proceeded during the following two years. Although some meetings were hosted at the sites of the different members throughout the country,

the bulk of the sessions were either at BEMA headquarters or at the IBM program development center in the Time-Life building, both in New York City.

The initial Fortran IV compilers were all under development while the work of X3.4.3-IV was in progress. The members of X3.4.3-IV were all either responsible or could direct changes in their language specifications for these implementations. It was a unique situation, where language changes adopted by the subcommittee were incorporated into the compilers almost immediately. I have always felt that the actual standardization of Fortran stemmed from the discussions, understandings, and agreements of X3.4.3-IV rather than from formal text that followed some years later.

The undocumented agreement within X3.4.3-IV was that the standard would not incorporate any feature that was not planned for all the implementations. Since the starting point for all of our language specs was the IBM-proposed language, it followed that the draft most closely represented the IBM implementation. It was by no means a slavish [unquestioning] copy. For one thing, there were no rigorous specifications within IBM of much of Fortran IV that could have been copied. This was particularly true in the input-output area. There were some features that IBM meant to carry into Fortran IV from their Fortran II implementations in order to protect their users' investment.

Unfortunately, some of their Fortran II implementations contained some objectionable shortcuts. For example, a constant could precede a variable and imply a multiplication operator (5L meant 5 * L). To their credit, there was never much of a hassle with those from IBM in deleting features that were objectionable carryovers from existing implementations of Fortran II. I believe they were sincerely motivated in working toward the best long-term interests of the language. Another change of note was that the DATA statement syntax was altered from the way IBM was implementing it. It was originally specified with parentheses rather than slashes as the delimiter for the list of constants.

Having no precedents, X3.4.3-IV had to address numerous problems common to all language standardization. Much of this we take for granted now, but there was nothing to turn to at the time. There were discussions as to whether there should be a standard. There is a penalty: The presence of a standard implies the pressure of conformance over a long period to a static document. This could certainly serve to limit the growth and development of the language. Even if motivated, the implementor, constrained to conform, would be prohibited

from adding extensions. Programs requiring nonstandard functionality could not be developed. Unanticipated requirements could not be satisfied until after the many years needed for a new revision had elapsed. The difficulties of specification of a standard could artificially limit the functionality because it might be too difficult or unwieldy to word the true restrictions. Once a feature was standardized, its life would be semi-eternal even if the feature were a mistake. The result is that generally a very conservative posture is assumed in deciding what is to be included. The potentially useful but untested functionality usually doesn't make it. These are all penalties to be weighed against the advantages of portability and communication that standardization could provide.

A partial answer to these objections to having a standard was worked into the interpretation section of the standard and has been carried into all the subsequent revisions of Fortran standards. The standard is to be interpreted as permissive. That is, that the standard serves only to specify a part, not all, of the language. Anything not specified isn't unclean, bad, immoral, or even not kosher. It is simply not specified. Similarly, things that are prohibited are things that are simply uninterpreted when violated. A standard program must be limited to what is specified in order to conform, but the same is not true for a processor. A processor may provide array processing, but it must handle standard subscripting in the conforming manner. Thus, an experimental extension can be available in a standard processor. The processor must be able to properly interpret standard programs, but may also provide interpretation to a nonconforming program. The choice is then available to conform or not as the economics dictate. Some nonconformance is encouraged.

The subcommittee decided that the target audience for the standard would be compiler implementors or those on users' staffs who were the Fortran support experts. It was felt that this latter group were competent in being able to implement a compiler; so, in effect, there was just the implementor that characterized the audience. It was felt that the standard should specify the requirements for a standard conforming program rather than a compiler, but I don't believe this was apparent in the document.

The decision was made to use English rather than some metalanguage. This was in the belief that the description of the semantics was the difficult problem. Use of a metalanguage would not help there. A metalanguage was at best only

assisting in tackling the easiest part of the description. It was felt that its precision did not compensate for the need to become familiar with the added formality. Interestingly, the one most useful area that could have been served by a precise description using a metalanguage is the FORMAT statement. There was actually an error in the way it was specified in the standard. I am still unaware of a complete and precise description of that statement using some metalanguage.

There were many challenges to our ability to describe. CDC had proposed that the new logical IF be a two-way branch analogous to the arithmetic IF. This would have saved us much descriptive grief in handling the concept of a compound statement that had in this one place crept into the language. For example, we could no longer accurately state that every statement could have a label. It also led to an unduly harsh restriction prohibiting some forms of the logical IF from being the terminal statement of a DO loop.

The greatest challenge to our descriptive capabilities was presented by the extended range of a DO loop. (There are some who would claim that this honor should go to the concept of second-level definition.) All the implementations of Fortran IV being developed allowed a more liberal extended range than the one appearing in the standard. The committee would have been amenable to a less restrictive extended range if it could only have been appropriately described. Everyone tried at least twice. Any definition that included statements about the sanctity of the contents of index registers, although reflecting the real concern, was inappropriate. The definition finally adopted was an accurate subset of what everyone was providing. The definition was felt to be reasonably understandable. Those of you who have struggled with that definition and its prerequisite concept of *completely nested nest* might quibble about the description being reasonably understandable. This is only because you did not struggle with some of the descriptions that were rejected. This was certainly an instance where the ability to describe limited the technical content. I believe that there is some of this effect in most standards. It is deluding not to admit it.

There were a surprisingly small number of new terms that had to be coined. Terminology common to several manuals was preferred, since it would already be familiar. The (usually missing) rigorous definitions of these terms had to be developed. Among the newly coined terms were: *definition* and *undefinition*

and their related states of being *defined* or *undefined*; *reference* as applied to data and to procedures; and *intrinsic function*.

The term “intrinsic function” had its birth at a bull session during one of our meetings. We had been discussing the classification of functions, using the then customary terms *open* and *closed* functions. *Open* functions meant in-line code; *closed* meant some internal procedure. There was the concern that the absolute function *ABS*, generally thought of as the obvious prototype of in-line code, was no longer such when the argument was of complex data type. Further, the tightening techniques being developed for some codes might make it attractive to put more formerly closed functions in line, for greater speed. Besides, the terminology smacked of a particular implementation consideration. Lou Gatt piped up with the idea that the basic characteristic of these functions was that they were cast into or intrinsic to the processor, and that therefore we should call them *intrinsic functions*. So credit for this term belongs to Lou.

We were later to find that a subtle side benefit of our standards work was the widespread use of the terminology used in the standard. Our terminology was generally accepted and replaced the proliferation of some terms for certain actions and objects that were in use before without any rigorous and agreed-upon definitions.

The subcommittee gave some consideration to how to enforce the standard through use of acceptance procedures. Two hurdles caused us to turn away from further work in this area.

We realized that an exhaustive verification was not possible. It might be misleading to develop some partial verification package that might be construed as being total. Any such official package might be misused as a standard performance benchmark. The second hurdle was simply not having the manpower to do the work. It was hoped that market pressures would lead to some accepted verification means, but we didn’t have the resources.

The subcommittee X3.4.3-II drafting the specification based on Fortran II was even smaller than that of X3.4.3-IV. Their membership, as I recall, was

Jack Palmer, IBM, chairman
Irwin Boris, Honeywell
Charles Davidson, University of Wisconsin, 1620 Users Group
Don Laird, Penn State University

Bob Bruneile, Honeywell Users and NIH
Bernice Weizenhoffer, IBM
Robert Hux, RCA

Partly because their target was better defined, X3.4.3-II completed their work and the first draft Fortran standard almost a year before X3.4.3-IV finished. They were directed by X3.4.3 to keep the draft on hold until X3.4.3-IV had its draft ready. There was still the hope at that time that a compatible standard representing Fortran II and Fortran IV could be produced.

Subsequently, X3.4.3 decided that there should be a standard for the full language and a standard that was a proper subset of the full language. It was not possible to use the X3.4.3-II draft as the subset because of the number of totally incompatible differences between Fortran II and Fortran IV. The result was that the work of the X3.4.3-II was discarded. The subset was created by deleting text from the X3.4.3-IV draft. I hope that the draft produced by X3.4.3-II finds its way into the archives of Fortran history. Through no fault of its own, the effort of X3.4.3-II was not incorporated. Their work is historically significant in that it was the first completed draft of any language standard.

In October 1964, the two proposed draft standards were published in the Communications of the ACM. These were the first standards ever proposed for a programming language. They severely taxed the editing and approval mechanisms of ASA and BEMA. Draft standards up to then rarely needed more than a page of text and that page usually had room for the diagrams of the screw thread. The inability to rigorously check for conformance was shattering. It is little wonder that it took almost a year and a half before final approval was obtained in April 1966. The full language standard was designated ASA X3.9-1966 Fortran and the subset, ASA Basic Fortran.

Early in the standardization effort, the European Computer Manufacturers Association (ECMA) submitted a proposed draft of what they felt the full language should contain. Since they were separated from the developments in this country, their proposal fell between the Basic Fortran and the full Fortran. X3.4.3 voted to standardize on only two levels. When Fortran standardization was considered by the International Standards Organization, the ANSI form and content was chosen as the basis. The ECMA subset in ANSI form was added as the intermediate of three levels.

INTERPRETATIONS PERIOD (1967-1970)

Late in 1967, the then disbanded X3.4.3 was recalled primarily through the urging of the National Bureau of Standards. NBS, and in particular, Betty Holberton, was attempting to produce a Federal standard for Fortran. Betty's examination of the X3.9-1966 Fortran standard led her to submit a few dozen questions on interpretation. Other clarification inquiries were received from other sources. The Fortran group was revived as the only body that could authoritatively provide the clarifications. This process turned out to be more tedious and demanding than the standardization effort itself. Because we were dealing with an approved standard, not a single comma could be altered without going through the same long approval cycle. Interpretations had to be based on a rationale developed from the standard's actual wording and not from what even the original authors felt it should have been. Two interpretation reports were published, but they took over three years of meetings to produce. The difficulty of that interpretation effort has had its impact on the form of the standard for Fortran 77. Those who participated in both efforts took pains to carefully examine every phrase to reduce to a minimum the chance of misinterpretation.

By 1968 enough extensions had appeared in the more current implementations, to have the Fortran group appoint someone to study whether these extensions should be standardized. Frank Engel was selected as the one to conduct this study. Following Frank's report, in January 1969, the committee voted not to reaffirm X3.9-1966 when its review period came up, but to provide a new draft standard.

The committee had a succession of chairmen during this period. Bill Heising was replaced by Dick Ridgeway. Heising later returned as chairman prior to having Dennis Hamilton assume the position. In September 1970 Frank Engle assumed the chair and was to last throughout the development of Fortran 77. Frank's tenure, the longest of any chairman, ended in October 1977 when Jeanne Adams, the current holder, was appointed.

Fortran 77

By early 1970 the interpretation activity had had it. There were unresolved issues that could not be handled within the wording of X3.9-1966. They decided that since the standard had to be reviewed and replaced or reaffirmed by 1971, it would be more productive to abandon the clarification work and devote their

energy to a replacement. It is interesting that the most pessimistic schedule proposed at that time had the draft available by the end of 1971. The initial effort did not sharpen the ability to predict the time required to develop a standard.

Criteria and goals were drawn up for what would become Fortran 77. Their gist was to evolve the language, keep it approximately the same "size," and be sure that its efficiency features would not be impaired. It was important that the standard should be in a much more expository form and be meaningful to a larger and less knowledgeable audience. The form of the revision was chosen to be a single standard containing two subset levels. A later decision removed the intermediate subset. Because of the single standard approach, ASA X3.10-1966 Basic Fortran would be discarded (i.e., not reaffirmed).

They further voted that the new draft standard would be an evolutionary development that would not invalidate programs written in the language of the 1966 standards. This position was subsequently modified in two significant areas. The Hollerith data type was deleted because it was replaced by the more functional and machine independent character data type. The zero trip DO loop was specified. Actually, the control conditions for a zero trip DO were conditions that were nonconforming to the 1966 standard. However, since many implementations interpreted these conditions by executing the statements in the range once, many programs would have to be adjusted. There were objections, even though the issue related to programs that were technically not standard conforming.

Six years of effort went into Fortran 77. That standard represented work on over two hundred technical proposals from all over the world. The cost of the effort was in excess of two million dollars. The text was almost six times the size of X3.9-1966. While some very significant language additions are present, the expansion was largely attributable to the effort to make the document more understandable. The draft had a completely different organization than the 1966 standard. Emphasis was on clarity rather than compactness and non-redundancy. Extensive use was made of word processing, a concordance tool (KWIC), computer graphics, and direct transcription to hard copy and fiche facilities. The very extensive editing, consistency checking, and rewriting; and the distribution of the numerous interim drafts; were made possible only by some herculean efforts of the two editors, Lloyd Campbell and J. C. Noll. The editorial staff of ANSI was presented with a camera-ready copy of the draft for

publication, thus avoiding the errors that might have been introduced by an ANSI stage of processing.

The features of the draft standard were publicly presented by X3J3 members at the West Coast Fortran Forum held in Anaheim, California, in February 1976. The following month, the draft standard appeared in a special edition of SIGPLan Notices. An East Coast Fortran Forum was later held at the National Bureau of Standards in Gaithersburg, Maryland. Smaller groups of X3J3 members presented sessions on the new language standard at meetings of professional societies, user groups, and at conferences. The public review was initiated and comments were solicited.

During the period of public comment and review 289 responses consisting of 1225 pages were received. This was probably the largest outpouring to any proposed standard as of that time. It took almost a year for the committee to complete the responses. The number of public comments was evidence of the large, present, and continuing interest in the language and the understandability of the document. Despite the earlier extensive checking by the committee, there were a number of changes and corrections incorporated because of the comments.

The major issue, as measured by the volume of comments received, was to add some facility in support of structured programming. [See Dijkstra's letter to Editor (ACM), page <#>.] There were a significant number of preprocessors available that enabled Fortran programmers to develop programs using statements such as IF . . . THEN . . .ELSE, DO WHILE, DO UNTIL, CASE statements and the like. These preprocessors would convert the source into valid Fortran statements. There was a clear requirement to place some of the facility directly into the language. In responding, the committee felt that although some facility should be added, there were many syntactic variations and an insufficient experience basis to select and standardize many of the constructs. They took an appropriately conservative action of adding only the BLOCK IF constructs. This addition, as specified by Walt Brainerd [see note at end of this selection], provided most of the important capability requested. It avoided adding and being stuck with some of the other constructs such as DO UNTIL that are already falling into disuse because of superior forms.

The reaction of X3J3 to the structured programming requests is a good example of how a responsible committee should avoid an over-reaction that would prematurely add features that it would shortly regret. Unfortunately, there are

counterexamples in Fortran 77 such as the ENTRY statement and the alternate RETURN that should not have been included.

Approval of the standard came in April 1978. The official designation is American National Standard programming language Fortran X3.9-1978. In March 1980, an International Standards Organization Fortran, based upon the ANSI standard and known as ISO 1539-1980, was approved by twenty-one countries. This document is essentially a cover that references ANS X3.9-1978 for the English text and the French standard NF Z65-110 for the French text. In September 1980, the US Federal Standard for Fortran (FIPS PUB 69), incorporating by reference X3.9-1978, was approved.

Next Revision (1978-Present [1982])

Following the approvals of the Fortran 77 standards, the expected lull in the standardization activity did not materialize. There was pressure to consider the additions received during the public response to Fortran 77 that were rejected as premature. New Fortran implementations were incorporating additions such as a free form for statements. CODASYL had established a group (Fortran Data Base Language Committee, FDBLC) to provide a foundation for the addition of a major database augmentation to the language. ISA and the Purdue Workshop had developed standards addressing issues of tasking, file synchronization, and event management. An interest in a graphic addition was looming.

The committee devoted its time during 1978 to the planning for the future direction of the language. They solicited the thoughts of many other interested groups such as ISA, CODASYL, IEEE, and SIGNUM who were known to be interested in Fortran extensions. The level of interaction with international bodies was dramatically increased. International meetings under the informal structure of ISO Fortran Experts Group were convened in Europe during 1977, 1978, 1979, and 1980. All of this activity was in the attempt to obtain a broad basis of experience upon which to develop the successor standard.

X3J3 felt confident it could manage desired additional language features such as free form for statements, new control and data structures, and even most of the array handling. They even felt comfortable in handling the removal of some of the basic restrictions such as dynamic storage allocation, recursion, identification via storage association, and storage related precision. However, they were unsure of how to cope with major augmentations such as the database and

graphics handling. The additions would be expensive, not only in the cost of the processors, but in the breadth of the language that would be impacted. Even those not interested in these features would be paying a price in terms of what they would have to know to work with the language. The committee knew it did not have the expertise to select among the competing forms of database and graphics facilities. It wanted to be able to responsibly control these augmentations and yet didn't see how a single committee could commandeer all of the expertise needed for this development and management.

The answer is one that is still evolving and is a change in the architecture of the language. It is called the core-plus-modules approach. The plan for the language revision, called Fortran 8X, is to specify a relatively small, general purpose, self-sustaining core language. There would be added features that would modernize and streamline the language. The size of this core language would not exceed that of Fortran 77 because there would be compensating deletions. The core would be provided with very strong facilities to be able to interface with modules whose use could be selectively chosen. These modules would have to follow some broad conventions established by the committee to qualify as part of the Fortran family.

There would be two classes of modules, language extension modules and application modules. A language extension module would be developed by X3J3 and would represent features that exceed the general purpose scope of the core. It might also consist of features that were desirable for addition, but that had not been subject to sufficient implementation or usage experience. An extension module could not be modified and approved for standardization without reconsideration of the core and all of the other language extension modules.

One special language extension module would be called the Obsolete (Transition) Features Module. This module would contain all of the features needed for compatibility with the previous revision (Fortran 77). Features being dropped in a revision would survive for one cycle in this module. When this module was employed, it would override any incompatible features of the current language.

An applications module would probably be specified by some group external to X3J3 and would address features of some special domain. Examples might be one (or more) of the database facilities, a query capability, or a graphics addition. These would probably take the form of a collateral standard, so its

maintenance could be managed independently. The hope is that through use of modularity, the heart of what is identified as Fortran might remain small.

FUTURES

Over this period of twenty years of standardization we have been through two complete cycles and are in the midst of a third. How long does this go on and when does it end? Jean Sammet once asked me if it weren't time for the Fortran gurus to get together and call an end to the effort so people can get on with the using of the good languages. I have reservations over which of the current choices should be crowned the "good" languages. There should be something fundamentally different and better to justify dropping the huge investment in the current languages. The replacement should have features that defy compatible inclusion in what we have.

Until this revolutionary development makes its appearance, interest in Fortran will remain. There is the story of the farmer who was asked by one of his eager turks why he didn't replace his old burro with one of the younger, sleeker, more highly tuned and spirited steeds. He looked at the young hand with wrinkled and wizened eyes and said, "When you have something yeh gotta be sure gets done, yeh goes with what you knows." So be it with Fortran.

REFERENCES

1. Heising, William P., and Richard K. Ridgeway. "Fortran." Proposal distributed to ASA X3.4.3, June 1962.

2. Heising, William P. "History and Summary of Fortran Standardization Development for the ASA. "Commun. of ACM (Vol. 7, No. 10) October 1964, 590.

3. ASA X3.4.3. "Fortran vs. Basic Fortran." Commun. of ACM (Vol. 7, No. 10) October 1964, pp. 591-625.

4. ASA. American Standard Fortran (ASA X3.9-1966).

5. ASA. American Standard Basic Fortran (ASA X3.10-1966).

6. USASI. "Clarification of Fortran Standards – initial progress. "Commun. of ACM (Vol. 12, No. 5) May 1969, pp. 289-294.

7. ANSI. "Clarification of Fortran Standards – second report." Commun. of ACM (Vol. 14, No. 10) October 1971, pp. 628-642.

8. Greenfield, Martin N. “Fortran – A History of a Pragmatic Language,” HLSUA 1974 Meeting, June 11, 1974.

9. Greenfield, Martin N. “Background and Interpretation of the Fortran Draft Proposed Standard.” The WEST COAST Fortran FORUM Anaheim, California, February 9, 1976.

10. ANSI. “Draft Proposed ANS Fortran.” SIGPLan Notices (Vol. 11, No. 3), March 1976.

11. Brainerd, W. editor. “Fortran 77.” Commun. of ACM (Vol. 21, No. 10), October 1978, pp. 806 – 820.

12. ANSI. American National Standard programming language Fortran, ANSI X3.9-1978.

13. ISO. Programming languages – Fortran. ISO 1539-1980.

14. US Department of Commerce National Bureau of Standards. Fortran. FIPS PUB 69. September 4, 1980.

15. CODASYL FDBLC. Fortran Data Base Facility, Journal of Development, January 1980

LPM Note:

“Fortran 77: featuring structured programming” (*Meissner and Organick*, 1980, p 480) describes the statements IF (e) THEN, ELSE IF (e) THEN, ELSE, and END IF.

A “block IF construct” begins with an IF-THEN statement and ends with an END IF statement. “Between the IF-THEN and the corresponding END IF there may appear any number of ELSE IF-THEN statements, and at most one ELSE (which must not precede any of the ELSE IF-THEN statements). Groups of statements [block IF constructs] delimited by IF-THEN and END IF must be properly nested, both with respect to other such groups and with respect to DO loops. Transfer of control into such groups is prohibited.”

Edsger W. Dijkstra (1968)
PolyTechnical University, Eindhoven, The Netherlands

Letters to the Editor

Go To Statement Considered Harmful

Communications of the ACM, Vol 11 (3), 1968 pp. 147-148

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of *go to* statements in the programs they produce. More recently I discovered why the use of the *go to* statement has such disastrous effects, and I became convinced that the *go to* statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish *the* desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo the process until the very same point?) If the program text is a pure concatenation of, say, assignment

statements (for the purpose of this discussion regarded as the descriptions of single actions) it is sufficient to point in the program text to a point between two successive action descriptions. (In the absence of *go to* statements I can permit myself the syntactic ambiguity in the last three words of the previous sentence: if we parse them as "successive (action descriptions)" we mean successive in text space; if we parse as "(successive action) descriptions" we mean successive in time.) Let us call such a *pointer* to a suitable place in the text a "textual index."

When we include conditional clauses (*if B then A*), alternative clauses *(if B then* A_1 *else* A_2*)*, choice clauses as introduced by C. A. R. Hoare (*case[i] of (*A_1, A_2, … , A_n*))*, or conditional expressions as introduced by J. McCarthy (B_1 -> E_l, B_2 -> E_2, . . . , B_n -> E_n*)*, the fact remains that the progress of the process remains characterized by a single textual index.

As soon as we include in our language procedures we must admit that a single textual index is no longer sufficient. In the case that a textual index points to the interior of a procedure body the dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like, *while B repeat A* or *repeat A until B*). Logically speaking, such clauses are now superfluous, because. we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.

Why do we need such independent coordinates? The reason is – and this seems to be inherent to sequential processes-that we can interpret the value of a variable only with respect to the progress of the process. If we wish to count the number, n say, of people in an initially empty room, we can achieve this by increasing n by one whenever we see someone entering the room. In the in-between moment that we have observed someone entering the room but have not yet performed the subsequent increase of n, its value equals the number of people in the room minus one!

The unbridled use of the *go to* statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the values of some well-chosen variables, but this is out of the question because it is relative to the progress that the meaning of these values is to be understood! With the *go to* statement one can, of course, still describe the progress uniquely by a counter counting the number of actions performed since program start (viz. a kind of normalized clock). The difficulty is that such a coordinate, although unique, is utterly unhelpful. In such a coordinate system it becomes an extremely complicated affair to define all those points of progress where, say, n equals the number of persons in the room minus one!

The *go to* statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program. One can regard and appreciate the clauses considered as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs, but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

It is hard to end this with a fair acknowledgment. Am I to judge by whom my thinking has been influenced? It is fairly obvious that I am not uninfluenced by Peter Landin and Christopher Strachey. Finally I should like to record (as I remember it quite distinctly) how Heinz Zemanek at the pre-ALGOL meeting in early 1959 in Copenhagen quite explicitly expressed his doubts whether the *go*

to statement should be treated on equal syntactic footing with the assignment statement. To a modest extent I blame myself for not having then drawn the consequences of his remark.

The remark about the undesirability of the *go to* statement is far from new. I remember having read the explicit recommendation to restrict the use of the *go to* statement to alarm exits, but I have not been able to trace it; presumably, it has been made by C. A. R. Hoare. in [l, Sec. 3.2.1.] Wirth and Hoare together make a remark in the same direction in motivating the case construction: "Like the conditional, it mirrors the dynamic structure of a program more clearly than *go to* statements and switches, and it eliminates the need for introducing a large number of labels in the program."

In [2] Giuseppe Jacopini seems to have proved the (logical) superfluousness of the *go to* statement. The exercise to translate an arbitrary flow diagram more or less mechanically into a jumpless one, however, is not to be recommended. Then the resulting flow diagram cannot be expected to be more transparent than the original one.

REFERENCES:

1. Wirth, Niklaus, and Hoare, C.A. R. A contribution to the development of ALGOL. Comm. ACM 9 (June 1966), 413-432.
2. Böhm, Corrado, and Jacopini, Guiseppe. Flow diagrams, Turing Machines and languages with only two formation rules. Comm. ACM 9 (May 1966), 366-371

Edsger W Dijkstra
Technological University
Eindhoven, The Netherlands

ANS X3.9 Fortran Revision – Final Report

Frank Engel, Chairman

American National Standards Institute Technical Committee X3J3 completed work on the revised American National Standard X3.9 Fortran [Fortran 77] in accordance with established procedures. By a nearly unanimous vote of 28 to 1, X3J3 recommends that X3 proceed with the final processing of the amended [Board of Standards Review] BSR 13.9 Fortran, document X3d3/90, as the ANS X3.9 Fortran (revised). Upon final approval, this document will supersede ANS X3.9-1966 Fortran [Fortran 66] and will provide definitions for both the Fortran language and the subset Fortran language. X3J3 also recommends that the ANS X3.10-1966 Basic Fortran be withdrawn.

A previous report, document X3/75-99, presented to X3 in October, 1975, described in detail the work of X3J3 in preparing the dpANS X3.9 Fortran document X3J3/75. This supplementary report covers the activities of X3J3 from December,, 1975 to the present, and includes the preparation and publication of BSR X3.9 Fortran (X3J3/76) for public review and comment, the processing of the public comments and responses, and the repair of document X3J3/76 that resulted from consideration of the public comments.

Publication of dpANS X3.9 Fortran

Following X3J3 letter ballot approval for submittal for further processing and X3 approval for publication for public review, the dpANS X3.9 Fortran full language and its subset was published as the March, 1976 issue of the ACM SIGPLan Notices, Vol.11, No.3. Of the initial printing of 8000 copies, 6000 went to SIGPLan members throughout the world; 400 copies were purchased by NBS for federal government use; 200 copies were distributed by CBEMA, including 85 sent to the International Standards Association; 100 copies were sold to Fortran Forum attendees; and 50 copies were purchased by the British Computer Society Fortran Specialists Group (BCS/FSG) for distribution in England. ACM reported this to be one of their most active publications, and that a second printing was necessary to meet the demand. Thus there have been over 8000 copies of the BSR X3.9 Fortran document distributed. In contrast, the COBOL document BSR X3.23 had a printing of 2000 copies, 1000 of which were purchased by a single vendor, 100 by the federal government, and 500 by the general public; and the PL/1 document

BSR X3.53 had a printing of 1000 copies, 300 going to the federal government and 400 to the general public. The price of the BSR X3.9 Fortran document was $5.00 per copy, while the COBOL and PL/1 documents sold for $6.00 and $8.00, respectively.

The document, including the cover, was printed from photo-ready copy prepared by J. Crawford Noll of Bell Telephone Laboratories, using automated text editing facilities that were available to X3J3 in the development of the draft proposed standard. The publication format featured a side-by-side presentation of the full Fortran language and the subset in a section by section alinement. The BSR X3.9 comprises 18 sections, six appendices, a table of contents and an index for a total of 200 pages.

Informing the Public

At the instigation of X3J3 two special meetings were organized to discuss the BSR X3.9 Fortran. The West Coast Fortran Forum sponsored by the Los Angeles Chapters of the ACM and SIGPLan was held in Anaheim preceding the Computer Science Conference in February 1976, and the ACM SIGPLan Fortran Forum III was held at the National Bureau of Standards in Gaithersburg in March, 1976. Both forums were co-sponsored by NBS, and programs featured X3J3 members giving presentations on various aspects of the proposed revision of the Fortran standard. Each forum was attended by over 200 persons who enthusiastically responded favorably to the proposed standard.

X3J3 members have made presentations on the proposed standard at other national meetings including the 1976 National Computer Conference, the 1976 ACM National Conference and the major user groups, and at local meetings such as ACM chapters and university colloquia. They have also discussed the proposed standard at international meetings including ECMA/TC8, SEAS, the BCS/FSG, the Purdue Workshop Fortran Committee and the IFIP Working Group 2.5 on Numerical Software.

Public Review and Comment Processing

The public review period began March l, 1976 with the publication of the SIGPLan Notices; however, due to administrative anomalies beyond the control of X3J3 the formal official ANSI announcement was delayed, and the closing date of the public review period was extended by X3 from July 4, as originally stated

in the published document, to September 28, 1976. Thus, the document was available for seven months for review by the general public.

The wide distribution and discussion of the document resulted in a large number of comments. Some 289 individual s and organizations submitted 1225 pages of comments, citing 2397 items to the attention of the committee. While many of the comment letters cited only one item (most notably the fifty-three that ignored everything, except the IF-THEN-ELSE proposal that had been publicized in FORWORD), there were a substantial number that cited more than twenty to thirty items each. Many letters indicated a very careful review and understanding of the document by the authors, and a sincere concern for the development of the Fortran language. The comments were overwhelmingly favorable and complimentary of the committee's efforts in revising the Fortran language standard, as exemplified by the following quotation from C143:

> "… Despite the extent of these comments and suggestions, my overall reaction is favorable and I would recommend its (BSR X3.9) adoption, even if none of my suggestions is acted upon…. Let me compliment X3J3 on a job well done." Then, after 60 pages of 74 comment items, he concluded with: "The proposed standard is such an improvement over the 1966 standard that I cannot wholeheartedly support any proposal, however valuable, which may delay its adoption."

A summary and analysis of the public comments is given in Attachment A. Some of the comments, anticipating the continued enthusiasm and interest in Fortran that will result from the adoption of this proposed standard, offered suggestions for inclusion in the *next* revision of the standard. Others suggested that new features be included in this current revision, or that some of the features be modified or deleted from the current revision. The comment items were distributed among several categories as follows:

Addition of new features 40%

Modifications to existing features 16%

Deletion of features 6%

Textual clarity, style, etc. 27%

Clarification & interpretation 6%

Misunderstanding 2%

The very few adverse comments received fell into several categories:

1. “Fortran is an ill-structured language and should not be extended.” – C213.2

2 “It fails utterly to correct any of the manifest failings of Fortran.” – C 213.16

3 “The standards committee does a disservice by removing such features” that invalidate existing standard conforming programs. – C25

4“The proposal does not go far enough …” – C213.4, C218

5 The proposed standard goes too far. “The committee should not do development work.” – C282

With the exception of the first critique, these represent diametrically opposing views that are impossible to satisfy simultaneously. The amended BSR X3.9 Fortran represents a good compromise between the extreme positions, between modest growth and development of the standard Fortran language and the state-of-the-art that is consistent with the criteria X3J3 established to govern this revision (cf X3/75-99). The first criticism likewise cannot be satisfied by any repair of the document. For everyone holding this position there are hundreds – perhaps even thousands – who feel that a new Fortran standard is needed.

At its July, 1976 meeting X3J3 began processing the comments. As each letter was received the X3J3 Secretary assigned it a sequence number, identified each separate item discussed in the letter, and assigned responsibility for each item to one of seven X3J3 working groups. The annotation C149.8-2 identifies the eighth item of comment letter C 149, for which working group 2 had cognizance. The annotated comment letters were distributed to the committee members as working document X3J3/81, with an index and cross reference list of group assignments by comment item and author. The members assigned to the seven working groups and the sections of the proposed standard for which each group was responsible are given Appendix E of the Minutes of the 54th and subsequent meetings.

The working group prepared an appropriate response to each comment item that was assigned to it. When a change to the BSR X3.9 document was deemed necessary by the working group, the group prepared and submitted a proposal for consideration by the full committee, together with the textual modifications recommended by the group to effect the change. The individual comment responses, containing a list of the comment items to which each applied, were distributed to each X3J3 member in working document X3J3/82. Any member could raise ob-

jection to a response and call for full committee consideration of any group response. Using text editing facilities provided by Tom Gibson and Bruce Puerling of Bell Telephone Laboratories, X3J3/8? was reissued as working document X3J3/82.1 in which the responses were reordered and arranged by comment letter and item number, instead of the response number organization employed in /82. Prior to the May, 1977 X3J3 meeting, each comment letter and its responses were assigned to at least three X3J3 members for an independent review and critique as to the relevance and adequacy of the response to each item. At the May, 1977 meeting the full committee then considered the results of the members' critiques, and approved the set of responses, as appropriately modified, that is contained in document X3J3/91.

For the record, documents X3J3/81 and /91 are transmitted herewith, and they may be examined by anyone desiring so to do. Presumably arrangements may be made with the X3 Secretariat, should anyone require copies of these voluminous documents for himself. X3J3 is sending to each commenter a copy of his annotated letter from X3J3/81 and the set of responses to his comments from X3J3/91.

Repair and Modification of the BSR X 3.9 Document

In considering the proposals arising from the public comments, X3J3 was reluctant to make any changes to the BSR X3.9 Fortran document. In particular, the committee wished to avoid any drastic changes that might necessitate another extended public review and further delay the promulgation of this overdue revision. Thus, a major reorganization and rewriting of certain sections was rejected, as was any change in style or formalism of syntactic description. Changes that corrected errors and misstatements or improved the clarity of the text were considered to be mandatory. The committee also recognized the desirability of modifying or removing syntactic forms that might inhibit the future development of the Fortran language as suggested by the public comments and/or that would improve program portability. Each proposed change to the BSR X3.9 Fortran document, whether editorial or substantive, was acted upon by the full committee at one of the nine meetings held since December, 1975. Most of the proposals were put forward by the working groups or by the editor, and a small number were offered independently by individual X3J3 members. Over 500 motions to change the document were considered, with 83% being approved, 14% rejected and 3% tabled without a deciding vote, these motions and proposals are fully documented in the several minutes of X3J3 meetings.

With respect to the several categories of the public comment items, the committee actions were as follows:

Addition of new features: 18% accepted

Modifications to existing features: 46% accepted

Deletion of features: 21% accepted

Textual cl arity, style, etc. 56% accepted

It would be noted that in rejecting a new feature for inclusion in this revision, X3J3 was not necessarily opposed to the feature as a part of the Fortran language, but the committee was deferring the feature until the next revision, rather than delay the promulgation of this revision. The committee had previously considered most of these items at one time or another and failed to complete the work necessary for their adoption. For some items more time is needed to develop a consensus, and experience to be gained with the implementation and use of this revision will be helpful in determining that consensus. Other items require much work and time to incorporate them into the document, and the interactive effects of such extensive changes militates against their being undertaken at this stage of the revision process. Similarly, in refusing to delete a feature from the language at this time, the committee was not precluding that from being done at some future time when another revision is undertaken.

Attachment B is a summary of the X3J3 actions with respect to substantive issues affecting the language features that arose from the consideration of the public comments. In addition to those actions that resulted in changes from X3J3/76, (BSR X3.9) document, Attachment B also includes a list of X3d3 actions that rejected proposed modifications and affirmed the features as described in X3J3/76. Finally, Attachment B contains a list of the major differences in the revised ANS X3.9 Fortran, X3J3/90 from the ANS X3.9-1966 Fortran that it replaces.

Informing the Public – Continued

During the year since the BSR Fortran document was published and the initial public review, X3J3 has striven to keep the public informed of its actions. The current X3J.3 mailing list contains 240 addresses. Included in the mailing list are representatives of ECMA/TC8 and the British, Danish, Dutch, French, Japanese and Swedish national standards bodies. They have received notices of X3J3 meetings and the complete minutes of those meetings, which as noted above, fully document the committee’ s actions. Of the eight interim revisions to X3J3/76 that

have been prepared as working documents of the committee, four have been distributed to the full mail ing list, including X3J3/76. 7 which includes al l substantive revisions. Following each X3J3 meeting, a committee approved press release was issued announcing the significant actions that had been taken.

An X3d3 member, Dr. Loren Meissner of the University of California Lawrence Berkeley Laboratories, is Editor of FOR-WORD, a newsletter published by the Fortran Development Committee of the ACM SIGPLan. The FOR-WORD distribution includes the entire X3J3 mailing list, every comment letter author, and every Fortran Forum attendee, among others interested in Fortran. Summaries of the changes to X3Ö3/76 that were approved by X3J3 have been regularly published in FOR-WORD by Dr. Meissner. These summaries also appeared in SIGPLan Notices in January, and April, 1977.

There has been an ongoing dialogue with the general public through continued correspondence and in person. Twenty-nine visitors have attended the nine X3J3 meetings and have interacted directly with the committee Following the Fortran Forums, X3J3 members met with interested attendees discussing the proposed standard and exchanging ideas about the language development. X3J3 members recently attended the BCS/FSG and ECMA/TC8 meetings to discuss the final amended BSR X3.9 Fortran. Representatives of those groups have attended X3J3 meetings, and Mr. Watson of JCL has become a member of X3J3 representing ECMA/TC8.

The reply to each comment letter in addition to the responses to each item per X3J3/91, will also include a copy of Attachment B, so that the author will be informed of all of the changes accepted and rejected by the committee, and not just the committee’s reaction to those items mentioned in his own letter.

Administrative Summary

X3/75-99 reported on the X3J3 meetings, membership participation and sponsor representation through October, 1975. Since that time there have been nine more committee meetings and further changes in the membership. Of the six members who then had served for eight years from the inception of the clarification and revision work in 1967, there remain four who now have completed ten years of active service with the committee. Due to reassignment of responsibilities within their sponsoring organizations, Carl Bailey of Sandia Corporation and Ward Klein of IBM have withdrawn from membership. Martin Greenfield of Honeywell Information Systems is vice chairman of X3J3 and was a member of ASA X3.4.3

committee and participated actively in the development of the 1966 standards. Lloyd Campbell of the U. S. Amy Ballistics Research Laboratory is secretary of X3J3, and as editor has been personally responsible for the consistency of style, grammar, etc. apparent in the document. Betty Holberton of the NBS and the writer are the other long standing members still on the committee, and together these four account for over twenty percent of the total meeting man years that have gone into the preparation of the document. There has been a slight reduction in representation of producers for small dp systems, and an increase in the participation by both government and general users of the Fortran language. These are reflected in the data of Table 1, which includes the information reported previously in X3/75-99.

TABLE 1: X3J3 MEETING SUMMARY

Year	Sponsor Representation				Avg. No. of V. M.*	No. of Meetings	Avg. Days/ Mgt.	Avg. Attendance	Mtg. Man-days**
	Producers		Users						
	Large	Small	Gen'l	Gov't					
1967	5	0	3	2	10	5	1	10	65
1968	6	0	5	2	12	5	2	9	90
1969	6	0	9	3	12	6	3	17	306
1970	6	0	10	3	21	6	3	18	324
1971	7	0	10	3	23	6	3	20	360
1972	6	2	13	6	24	6	3	22	396
1973	7	3	13	5	23	6	4	23	552
1974	7	5	9	4	27	6	4	23	552
1975	7	6	9	3	25	8	5	21	725
1976	7	4	9	5	25	5	4	21	399
1977	7	3	12	6	28	3	5	25	358***

* Voting Member
** Meeting-Man-Days
***This Table includes Through May 1977

Since October, 1975, X3J3 members expended 852 meeting man days in completing its letter ballot responses, preparing X3J3/76 for publication as BSR X3.9 Fortran, considering comments, preparing and approving comment responses, and repairing the document for submission for final approval as the ANS X3.9 Fortran. Thus, the total effort invested in producing the amended document which we now

submit for your approval is at least 33 man years, 6.8 man years representing the latter period covered by this report. As suggested in X3/75-,99, this represents an expenditure in excess of two million dollars. The text editing resource, document preparation and distribution represents an additional expenditure in excess of $100,000.

Future Projections

There is a strong consensus in X3J3 that the amended BSR X3.9 Fortran is a good and sound base upon which future Fortran standard developments can take place in whatever direction they may go. With the cooperation of X3 and ANSI, these events will take place:

June 1977: X3 acceptance of our recommendation to proceed with the final processing of the revised standard

July to September 1977: The 60-day X3 letter ballot for final approval and concurrent public review period

October 1977: X3J3 consider and respond to those issues that may arise from the letter ballot and final public review, if any

November 1977: Publication of ANS X3.9-1977 Fortran

November 1977: International Standards Organization TC97/SC5 Fortran working group having requested and received document X3J3/90, meeting to consider its adoption as revision of ISO R-1539 Fortran. The AFNOR organization has begun the translation of BSR X3.9 into French, and has requested that no more changes be made

November 1980: X3J3 publication of dpANS X3.9 Fortran for the next revisionuse

November 1982: Publication of ANS X3.9-1982 Fortran

From the consideration of the public comments on BSR X3.9 Fortran, it is apparent that another revision of the standard should be anticipated. The latest date for it to appear should be November 1982, if the five year ANSI review cycle is honored, and the current revision is issued in November 1977 as postulated. Assuming that two years will be required for publication, public review and comment, and final processing (as occurred this time), then the draft proposed next revision document should be completed in November, 1980. That being scarcely three years away, it is essential that X3J3 immediately initiate the preparation of that document.

In October, 1975, I appointed a subcommittee on future revisions, and directed that it:

> Develop suitable criteria to determine whether or not another revision of ANS X3.9 is necessary and desirable
>
> Develop suitable objectives for a future revision of ANS X3e9
>
> Develop suitable criteria for acceptance or rejection of proposed changes to the language
>
> Determine how the language can best adapt to future needs and divers requirements to collect and classify candidate features for a future revision

Because of the ongoing work on the current revision and the demands on X3J3 resources to respond to the public comments, this subcommittee will hold its first meeting in July. It will draft the SD-3 document proposing that a project be established to. prepare a draft proposed revision of ANS X3.9-1977 Fortran for release for public review and comment in 1980. It has taken two years to change the attitude of X3J3 members from trying to get one more new feature into this revision, to focusing on the next revision as the proper time to make other needed extensions to the language. A not insignificant contributing factor in this transition has been sponsor pressure to terminate this project that has extended over ten years. Sponsors, too, have been looking at the cost in individual terms, and have set limits for their continued participation.

The alternative to this projected schedule of events, should further repair of BSR X3.9 Fortran be contemplated, is not at all attractive, unless one' s objective is to forestall the continued popularity of Fortran. To amend this document again would mean at least another two years' delay in its final adoption, inasmuch as that is the time it takes to go through the cycle of repair, publish, respond, vote, etc. Opening up for reconsideration, with the anticipated change in X3J3 membership, will most certainly result in changes to the language, not all of which will be those desired by the dissenting voices. Such changes made under the duress of a short „review and repair" cycle can only degrade the document, as a standard, rather than improve it. However, in terms of the time frame, this amended, inferior document would appear in November 1979, just one year before the draft proposed ANS X3.9 for the following revision will appear, according to our projected schedule of events. There is no assurance that the amended amended document will be any more acceptable to all who then ballot. Could it be that the cycle would repeat again, and again, and …?

There are strong indications that the world will not wait another two years for an acceptable Fortran standard. BSR X3.9 Fortran, either in the published version of X3J3/76 or the amended version of X3J3/90 may very well become the de facto standard without becoming the “approved” ANSI standard. We are aware of at least two existing implementations for significant systems that profess to be “Fortran 77 conforming processors,” and there are indications that most producers are well along with implementations of such processors for their systems. There have been rumors that there is pressure for the federal government to adopt a federal standard for Fortran; If there is not an ANSI standard adopted now, then’ that federal standard would be the BSR X3.9 Fortran. We believe that a further delay in adopting the revised Fortran standard will not be in the best interests of standardization, but will encourage the proliferation of those practices that standardization is intended to avoid.

Conclusion

The American National Standards Institute Technical Committee X3J3 has fully and responsibly discharged its obligations:

To produce the revised ANS X3.9 Fortran

To inform the public of all its actions

To respond to the needs of the Fortran community throughout the world

The public has responded enthusiastically and overwhelmingly in favor of the committee’s actions, and eagerly awaits the new standard. The international standards body is prepared to proceed immediately with the new American National Standard X3. 9-1977 Fortran as the basis for revision of ISO R-1539 Fortran. In response to public demand, X3J3 has initiated work leading to the preparation of the draft proposed revision document to meet the next mandatory revision cycle.

Numerical Recipes in Fortran 90

Second Edition, Volume 2, of Fortran Numerical Recipes
CAMBRIDGE UNIVERSITY PRESS, 1996

FOREWORD

(Pages x-xx)

Michael Metcalf

Sipping coffee on a sunbaked terrace can be surprisingly productive. One of the Numerical Recipes authors and I were each lecturing at the International Center for Theoretical Physics in Trieste, Italy, he on numerical analysis and I on Fortran 90. The numerical analysis community had made important contributions to the development of the new Fortran standard, and so, unsurprisingly, it became quickly apparent that the algorithms for which Numerical Recipes had become renowned could, to great advantage, be recast in a new mold. These algorithms had, hitherto, been expressed in serial form, first in Fortran 77 and then in C, Pascal, and Basic. Now, nested iterations could be replaced by array operations and assignments, and the other features of a rich array language could be exploited. Thus was the idea of a "Numerical Recipes in Fortran 90" first conceived and, after three years' gestation, it is a delight to assist at the birth.

But what is Fortran 90? How did it begin, what shaped it, and how, after nearly foundering, did its driving forces finally steer it to a successful conclusion?

The Birth of a Standard

Back in 1966, the version of Fortran now known as Fortran 66 was the first language ever to be standardized, by the predecessor of the present American National Standards Institute (ANSI). It was an all-American affair. Fortran had first been developed by John Backus of IBM in New York, and it was the dominant scientific programming language in North America. Many Europeans preferred Algol (in which Backus had also had a hand). Eventually, however, the mathematicians who favored Algol for its precisely expressible syntax began to defer to the scientists and engineers who appreciated Fortran's pragmatic, even natural, style. In 1978, the upgraded Fortran 77 was standardized by the ANSI technical committee, X3J3, and subsequently endorsed by other national bodies

and by ISO in Geneva, Switzerland. Its dominance in all fields of scientific and numerical computing grew as new, highly optimizing compilers came onto the market. Although newer languages, particularly Pascal, Basic, PL/1, and later Ada attracted their own adherents, scientific users throughout the 1980s remained true to Fortran. Only towards the end of that decade did C draw increasing support from scientific programmers who had discovered the power of structures and pointers.

During all this time, X3J3 kept functioning, developing the successor version to Fortran 77. It was to be a decade of strife and contention. The early plans, in the late 1970s, were mainly to add to Fortran 77 features that had had to be left out of that standard. Among these were dynamic storage and an array language, enabling it to map directly onto the architecture of supercomputers, then coming onto the market. The intention was to have this new version ready within five years, in 1982. But two new factors became significant at that time. The first was the decision that the next standard should not just codify existing practice, as had largely been the case in 1966 and 1978, but also extend the functionality of the language through innovative additions (even though, for the array language, there was significant borrowing from John Iverson's APL and from DAP Fortran). The second factor was that X3J3 no longer operated under only American auspices. In the course of the 1980s, the standardization of programming languages came increasingly under the authority of the international body, ISO. Initially this was in an advisory role, but now ISO is the body that, through its technical committee WG5 (in full, ISO/IEC JTCl/SC22/WG5), is responsible for determining the course of the language. WG5 also steers the work of the development body, then as now, the highly skilled and competent X3J3. As we shall see, this shift in authority was crucial at the most difficult moment of Fortran 90's development.

The internationalization of the standards effort was reflected in the welcome given by X3J3 to six or seven European members; they, and about one-third of X3J3's U.S. members, provided the overlapping core of membership of X3J3 and WG5 that was vital in the final years in bringing the work to a successful conclusion. X3J3 membership, which peaked at about 45, is restricted to one voting member per organization, and significant decisions require a majority of two-thirds of those voting.

Nationality plays no role, except in determining the U.S. position on an international issue. Members, who are drawn mainly from the vendors, large research laboratories, and academia, must be present or represented at two-thirds of all meetings in order to retain voting rights.

In 1980, X3J3 reported on its plans to the forerunner of WG5 in Amsterdam, Holland. Fortran 8x, as it was dubbed, was to have a basic array language, new looping constructs, a bit data type, data structures, a free source form, a mechanism to "group" procedures, and another to manage the global name space. Old features, including COMMON, EQUIVALENCE and the arithmetic-IF, were to be consigned to a so-called obsolete module, destined to disappear in a subsequent revision. This was part of the "core plus modules" architecture, for adding new features and retiring old ones, an aid to backwards compatibility. Even though Fortran 77 compilers were barely available, the work seemed well advanced and the mood was optimistic. Publication was intended to take place in 1985. It was not to be.

One problem was the sheer number of new features that were proposed as additions to the language, most of them worthwhile in themselves but with the totality being too large. This became a recurrent theme throughout the development of the standard. One example was the suggestion of Lawrie Schonfelder (Liverpool University), at a WG5 meeting in Vienna, Austria, in 1982, that certain features already proposed as additions could be combined to provide a full-blown derived data type facility, thus providing Fortran with abstract data types. This idea was taken up by X3J3 and has since come to be recognized, along with the array language, as one of the two main advances brought about by what became Fortran 90. However, the ramifications go very deep: all the technical details of how to handle arrays of objects of derived types that in turn have array components that have the pointer attribute, and so forth, have to be precisely defined and rigorously specified.

Conflict

The meetings of X3J3 were often full of drama. Most compiler vendors were represented as a matter of course but, for many, their main objective appeared to be to maintain the status quo and to ensure that Fortran 90 never saw the light of day. One vendor's extended (and much-copied) version of Fortran 77 had virtually become an industry standard, and it saw as its mission the maintenance

of this lead. A new standard would cost it its perceived precious advantage. Other large vendors had similar points of view, although those marketing supercomputers were clearly keen on the array language. Most users, on the other hand, were hardly prepared to invest large amounts of their employers' and their own resources in simply settling for a trivial set of improvements to the existing standard. However, as long as X3J3 worked under a simple majority voting rule, at least some apparent progress could be made, although the underlying differences often surfaced. These were even sometimes between users – those who wanted Fortran to become a truly modern language and those wanting to maintain indefinite backwards compatibility for their billions of lines of existing code.

At a watershed meeting, in Scranton, Pennsylvania, in 1986, held in an atmosphere that sometimes verged on despair, a fragile compromise was reached as a basis for further work. One breakthrough was to weaken the procedures for removing outdated features from the language, particularly by removing no features whatsoever from the next standard and by striking storage association (i.e., COMMON and EQUIVALENCE) from the list of features to be designated as obsolescent (as they are now known). A series of votes definitively removed from the language all plans to add: arrays of arrays, exception handling, nesting of internal procedures, the FORALL statement (now in Fortran 95), and a means to access skew array sections. There were other features on this list that, although removed, were reinstated at later meetings: user-defined operators, operator overloading, array and structure constructors, and vector-valued subscripts. After many more travails, the committee voted, a year later, by 26 votes to 9, to forward the document for what was to become the first of three periods of public comment.

While the document was going through the formal standards bureaucracy and being placed before the public, X3J3 polished it further. X3J3 also prepared procedures for processing the comments it anticipated receiving from the public, and to each of which, under the rules, it would have to reply individually. It was just as well. Roughly 400 replies flooded in, many of them very detailed and, disappointingly for those of us wanting a new standard quickly, unquestionably negative towards our work. For many it was too radical, but many others pleaded for yet more modern features, such as pointers.

Now the committee was deadlocked. Given that a document had already been published, any further change required not a simple but a two-thirds majority. The conservatives and the radicals could each block a move to modify the draft standard, or to accept a revised one for public review – and just that happened, in Champagne-Urbana, Illinois, in [May] 1988. Any change, be it on the one hand to modify the list of obsolescent features, to add the pointers or bit data type wanted by the public, to add multi-byte characters to support Kanji and other non-European languages or, on the other hand, to emasculate the language by removing modules or operator overloading, and hence abstract data types, to name but some suggestions, none of these could be done individually or collectively in a way that would achieve consensus. I wrote:

"In my opinion, no standard can now emerge without either a huge concession by the users to the vendors (MODULE / USE) and/or a major change in the composition of the committee. I do not see how members who have worked for up to a decade or more, devoting time and intellectual energy far beyond the call of duty, can be expected to make yet more personal sacrifices if no end to the work is in sight, or if that end is nothing but a travesty of what had been designed and intended as a modern scientific programming language. … I think the August [1988] meeting will be a watershed — if no progress is achieved there will be dramatic resignations, and ISO could even remove the work from ANSI, which is failing conspicuously in its task." (However, the same notes began with a quotation from The Taming of the Shrew: "And do as adversaries do in law, / Strive mightily, but eat and drink / as friend." That we always did, copiously.)

Resolution

The "August meeting" was, unexpectedly, imbued with a spirit of compromise that had been so sadly lacking at the previous one. Nevertheless, after a week of discussing four separate plans to rescue the standard, no agreement was reached. Now the question seriously arose: Was X3J3 incapable of producing a new Fortran standard for the international community, doomed to eternal deadlock, a victim of ANSI procedures?

Breakthrough was achieved at a traumatic meeting of WG5 in Paris, France, a month later [September 1988]. The committee spent several extraordinary days drawing up a detailed list of what it wanted to be in Fortran 8x. Finally, it

set X3J3 an ultimatum that was unprecedented in the standards world: The ANSI committee was to produce a new draft document, corresponding to WG5's wishes, within five months! Failing that, WG5 would assume responsibility and produce the new standard itself.

This decision was backed by the senior U.S. committee, X3, which effectively directed X3J3 to carry out WG5's wishes. And it did [with some stretching of the "five months" deadline]! The following November [1989], it implemented most of the technical changes, adding pointers, bit manipulation intrinsic procedures, and vector-valued subscripts, and removing user-defined elemental functions (now in Fortran 95). The actual list of changes was much longer. X3J3 and WG5, now collaborating closely, often in grueling six-day meetings, spent the next 18 months and two more periods of (positive) public comment putting the finishing touches to what was now called Fortran 90, and it was finally adopted, after some cliff-hanging votes, for forwarding as a U.S. and international standard on April 11, 1991, in Minneapolis, Minnesota.

Among the remaining issues that were decided along the way were whether pointers should be a data type or be defined in terms of an attribute of a variable, implying strong typing (the latter was chosen), whether the new standard should coexist alongside the old one rather than definitively replace it (it coexisted for a while in the U.S., but was a replacement elsewhere, under ISO rules), and whether, in the new free source form, blanks should be significant (fortunately, they are).

Fortran 90

The main new features of Fortran 90 are, first and foremost, the array language and abstract data types. The first is built on whole array operations and assignments, array sections, intrinsic procedures for arrays, and dynamic storage. It was designed with optimization in mind.

The second is built on modules and module procedures, derived data types, operator overloading and generic interfaces, together with pointers. Also important are the new facilities for numerical computation including a set of numeric inquiry functions, the parametrization of the intrinsic types, new control constructs — SELECT CASE and new forms of DO, internal and recursive procedures and optional and keyword arguments, improved I/O facilities, and many new intrinsic procedures. Last but not least are the new free source form,

an improved style of attribute-oriented specifications, the IMPLICIT NONE statement, and a mechanism for identifying redundant features for subsequent removal from the language. The requirement on compilers to be able to identify, for example, syntax extensions, and to report why a program has been rejected, are also significant. The resulting language is not only a far more powerful tool than its successor [predecessor?], but a safer and more reliable one too. Storage association, with its attendant dangers, is not abolished, but rendered unnecessary. Indeed, experience shows that compilers detect errors far more frequently than before, resulting in a faster development cycle. The array syntax and recursion also allow quite compact code to be written, a further aid to safe programming.

No programming language can succeed if it consists simply of a definition (witness Algol 68). Also required are robust compilers from a wide variety of vendors, documentation at various levels, and a body of experience. The first Fortran 90 compiler appeared surprisingly quickly, in 1991, especially in view of the widely touted opinion that it would be very difficult to write one. Even more remarkable was that it was written by one person, Malcolm Cohen of NAG, in Oxford, U.K. There was a gap before other compilers appeared, but now they exist as native implementations for almost all leading computers, from the largest to PCs. For the most part, they produce very efficient object code; where, for certain new features, this is not the case, work is in progress to improve them.

The first book, Fortran 90 Explained, was published by John Reid and me shortly before the standard itself was published. Others followed in quick succession, including excellent texts aimed at the college market. At the time of writing there are at least 19 books in English and 22 in various other languages: Chinese, Dutch, French, Japanese, Russian, and Swedish. Thus, the documentation condition is fulfilled.

The body of experience, on the other hand, has yet to be built up to a critical size. Teaching of the language at college level has only just begun. However, I am certain that this present volume will contribute decisively to a significant breakthrough, as it provides models not only of the numerical algorithms for which previous editions are already famed, but also of an excellent Fortran 90 style, something that can develop only with time. Redundant features are abjured. It shows that, if we abandon these features and use new ones in their

place, the appearance of code can initially seem unfamiliar, but, in fact, the advantages become rapidly apparent. This new edition of Numerical Recipes stands as a landmark in this regard.

Fortran Evolution

The formal procedures under which languages are standardized require them either to evolve or to die. A standard that has not been revised for some years must either be revised and approved anew, or be withdrawn. This matches the technical pressure on the language developers to accommodate the increasing complexity both of the problems to be tackled in scientific computation and of the underlying hardware on which programs run. Increasing problem complexity requires more powerful features and syntax; new hardware needs language features that map onto it well.

Thus it was that X3J3 and WG5, having finished Fortran 90, began a new round of improvement. They decided very quickly on new procedures that would avoid the disputes that bedeviled the previous work: WG5 would decide on a plan for future standards, and X3J3 would act as the so-called development body that would actually produce them. This would be done to a strict timetable, such that any feature that could not be completed on time would have to wait for the next round. It was further decided that the next major revision should appear a decade after Fortran 90 but, given the somewhat discomforting number of requests for interpretation that had arrived, about 200, that a minor revision should be prepared for mid-term, in 1995. This should contain only “corrections, clarifications and interpretations” and a very limited number (some thought none) of minor improvements.

At the same time, scientific programmers were becoming increasingly concerned at the variety of methods that were necessary to gain efficient performance from the ever-more widely used parallel architectures. Each vendor provided a different set of parallel extensions for Fortran, and some academic researchers had developed yet others. On the initiative of Ken Kennedy of Rice University, a High-Performance Fortran Forum was established. A coalition of vendors and users, its aim was to produce an ad hoc set of extensions to Fortran that would become an informal but widely accepted standard for portable code. It set itself the daunting task of achieving that in just one year, and succeeded. Melding existing dialects like Fortran D, CM Fortran, and Vienna Fortran, and

adopting the new Fortran 90 as a base, because of its array syntax, High-Performance Fortran (HPF) was published in 1993 and has since become widely implemented. However, although HPF was designed for data parallel codes and mainly implemented in the form of directives that appear to non-HPF processors as comment lines, an adequate functionality could not be achieved without extending the Fortran syntax. This was done in the form of the PURE attribute for functions — an assertion that they contain no side effects — and the FORALL construct — a form of array assignment expressed with the help of indices.

The dangers of having diverging or competing forms of Fortran 90 were immediately apparent, and the standards committees wisely decided to incorporate these two syntactic changes also into Fortran 95. But they didn't stop there. Two further extensions, useful not only for their expressive power but also to access parallel hardware, were added: elemental functions, ones written in terms of scalars but that accept array arguments of any permitted shape or size, and an extension to allow nesting of WHERE constructs, Fortran's form of masked assignment. To readers of Numerical Recipes, perhaps the most relevant of the minor improvements that Fortran 95 brings are the ability to distinguish between a negative and a positive real zero, automatic deallocation of allocatable arrays, and a means to initialize the values of components of objects of derived data types and to initialize pointers to null.

The medium-term objective of a relatively minor upgrade has been achieved on schedule. But what does the future hold? Developments in the underlying principles of procedural programming languages have not ceased. Early Fortran introduced the concepts of expression abstraction (X = Y + Z) and later control expression (e.g., the DO loop). Fortran 77 continued this with the if-then-else, and Fortran 90 with the DO and SELECT CASE constructs. Fortran 90 has a still higher level of expression abstraction (array assignments and expressions) as well as data structures and even full-blown abstract data types. However, during the 1980s the concept of objects came to the fore, with methods bound to the objects on which they operate. Here, one particular language, C++, has come to dominate the field. Fortran 90 lacks a means to point to functions, but otherwise has most of the necessary features in place, and the standards committees are now faced with the dilemma of deciding whether to make the planned Fortran 2000 a fully object-oriented language. This could possibly

jeopardize its powerful, and efficient, numerical capabilities by too great an increase in language complexity, so should they simply batten down the hatches and not defer to what might be only a passing storm? At the time of writing, this is an open issue. One issue that is not open is Fortran's lack of in-built exception handling. It is virtually certain that such a facility, much requested by the numerical community, and guided by John Reid, will be part of the next major revision. The list of other requirements is long but speculative, but some at the top of the list are conditional compilation, command line argument handling, I/O for objects of derived type, and asynchronous I/O (which is also planned for the next release of HPF). In the meantime, some particularly pressing needs have been identified, for the handling of floating-point exceptions, interoperability with C, and allowing allocatable arrays as structure components, dummy arguments, and function results. These have led WG5 to begin processing these three items using a special form of fast track, so that they might become optional but standard extensions well before Fortran 2000 itself is published in the year 2001.

Conclusion

Writing a book is always something of a gamble. Unlike a novel that stands or falls on its own, a book devoted to a programming language is dependent on the success of others, and so the risk is greater still. However, this new Numerical Recipes in Fortran 90 volume is no ordinary book, since it comes as the continuation of a highly successful series, and so great is its significance that it can, in fact, influence the outcome in its own favor. I am entirely confident that its publication will be seen as an important event in the story of Fortran 90, and congratulate its authors on having performed a great service to the field of numerical computing.

Journal of Computer Science and Technology

Vol 11, No 1, Apr 2011

See also: Encyclopedia of Science and Technology, vol. 6
(Academic Press, 1992): Fortran (Pages 632-637)

The Seven Ages of Fortran

Michael Metcalf

Abstract

When IBM's John Backus first developed the Fortran programming language, back in 1957, he certainly never dreamt that it would become a world-wide success and still be going strong many years later. Given the oft-repeated predictions of its imminent demise. starting around 1968. it is a surprise. even to some of its most devoted users. that this much-maligned language is not only still with us. but is being further developed for the demanding applications of the future. What has made this programming language succeed where most slip into oblivion?

One reason is certainly that the language has been regularly standardized. In this paper we will trace the evolution of the language from its first version and though six cycles of formal revision. and speculate on how this might continue.

Now modern Fortran is a procedural imperative compiled language with a syntax well suited to a direct representation of mathematical formulas. Individual procedures may be compiled separately or grouped into modules. either way allowing the convenient construction of very large programs and procedure libraries. Procedures communicate via global data areas or by argument association. The language now contains features for array processing. abstract data types, dynamic data structures. object oriented programming, and parallel processing.

Language evolution

1. The First Age: Origins

In the early days of computing, programming was tedious in the extreme – every tiny step had to be coded as a separate machine instruction, and the programmer had to be familiar with the intimate details of the computer's operation. Spurred by a perceived economic need to provide a form of ‘automatic programming’ to allow efficient use of manpower and computers, Backus proposed, at the end of 1953, to begin the development of the Fortran programming language (the name being a contraction of FORmula TRANslation). The overriding objective of the development team was to produce a compiler that would produce efficient object code comparable to that of hand-written assembly code.

Fortran came as a breakthrough. Instead of writing some obscure hieroglyphics, say as an instruction to divide two variables A and B and to save the result in C, the programmer could now write a more intelligible and natural statement, namely

```
C = A/B
```

This is called *expression abstraction*, because mathematical expressions could be written more-or-less as they appear in a textbook. Herein lay the secrets of Fortran’s initial rapid spread: scientists could write programs to solve problems themselves, in a familiar way and with only limited recourse to professional programmers, and that same program, once written, could be transported to any other computer which had a Fortran compiler. The first version, now known as Fortran I, contained early forms of constructs that have survived to the present day: simple and subscripted variables, the assignment statement, a do-loop, mixed-mode arithmetic, and input/output (I/O) specifications.

Many novel compiling techniques had to be developed, and it was not until 1957 that the first compiler was released to users of the target machine, the IBM 704. First experience showed that, indeed, it increased programmer efficiency and allowed scientists and engineers to program easily for themselves. The source form and syntax liberated programmers from the rigid input formats of assembly languages. Fortran was an immediate success.

Ease of learning and stress on optimization are two hallmarks of Fortran that have contributed to its continued popularity.

Based on the experience with Fortran I, it was decided to introduce a new version, Fortran II, in 1958. The crucial differences between the two were the introduction of subprograms, with their associated concepts of shared data areas, and separate compilation. Fortran II became the basis for the development of compilers by other manufacturers. A more advanced version was developed for the IBM 704 – Fortran III – but it was never released.

2. The Second Age: Fortran 66

In 1961, an IBM users' organization requested from IBM a new version, now called Fortran IV, which contained type statements, the logical-if statement, the possibility to pass procedure names as arguments, and the data statement and block data subprogram. Some original features, such as device-dependent I/O statements, were dropped.

Fortran IV was released in 1962 and quickly became available on other machines, but often in the form of a dialect. Indeed, the proliferation of these dialects led an American Standards Association (ASA) Working Group to develop a standard definition for the language. In 1966, a standard for Fortran was published, based on Fortran IV. This was the first programming language to achieve recognition as a national, and subsequently international (ISO, Geneva), standard, and is now known as Fortran 66.

Fortran 66 was made available on almost every computer made at that time, and was often pressed into service for tasks for which it had never been designed. Thus began a period during which it was very popular with scientists, but newer, more modern languages were appearing, including Algol 60, whose 'superior' concepts led to predictions that it would rapidly replace 'old fashioned` Fortran because of the latter's limitations. It became increasingly criticized, especially by academic computer scientists.

3. The Third Age: Fortran 77

The permissiveness of the Fortran 66 standard, whereby any extensions could be implemented on a given processor so long as it still correctly processed a standard-conforming program, led again to a proliferation of dialects. These dialects typically provided much-needed additional features, such as bit handling, or gave access to hardware-specific features, such as byte-oriented data types. Since the rules of ASA's successor, the American National Standards

Institute (ANSI), required that a standard be reaffirmed, withdrawn or revised after a five-year period has elapsed, the reformed Fortran committee, X3J3, decided on a revision. This was published by ANSI, and shortly afterwards by ISO, in 1978, and became known as Fortran 77. The new standard brought with it many new features, for instance the `if…then…else` construct (from the push for 'structured programming'), a character data type, and much enhanced I/O.

The new language was rather slow to spread. This was due in part to certain conversion problems and also to the decision of one large manufacturer, IBM, not to introduce a new compiler until 1982. It was thus only in the mid-1980s that Fortran 77 finally took over from Fortran 66 as the most used version. Ultimately, it became a hugely successful language for which compilers were available on every type of computer from the PC to the mighty Cray. Programs written in Fortran 77 were routinely used to perform such diverse calculations as designing the shapes of airplane fuselages, predicting the structures of organic molecules, and simulating the flow of winds over mountains.

Algol is now a dead language, however it begat descendants, most notably Pascal and Ada, and these too, in their time, together with IBM's PL/1, were variously considered to be about to deliver the *coup-de-grâce* to Fortran. But the new standard lent it a new vigour that allowed it to maintain its position as the most widely used scientific applications language of the time. However, it began to yield its position as a teaching language.

The entire issue of the journal Annals of the History of Computing. Vol. 6, No. 1 (1984) is devoted to papers on the early history of Fortran.

4. The Fourth Age: The battle for Fortran 90

As computers doubled in power every few years, and became able to perform calculations on many numbers simultaneously, by the use of processors running in parallel, and as the problems to be solved became ever more complex, the question arose as to whether Fortran 77 was still adequate. (And there were a large number of user requests left over that it had not been possible to include in it.) Programs of over a million lines became commonplace, and managing their complexity and having the means to write them reliably and understandably – so that they produce correct results and could later be modified – were desperately required.

Thus began the battle over Fortran 90 [and a change to lower case spelling]. Fortran had been attacked by computer scientists on two grounds. One was because of its positively dangerous aspects, for instance the lack of any inherent protection against overwriting the contents of memory in the computer, including the program instructions themselves! The other was its lack of indispensible language features, such as the ability to control the logical flow through a program in a clearly structured manner. On the other hand, Fortran had always been a relatively easy language to learn and that, combined with its emphasis on efficient, high-speed processing, had kept it attractive to many busy scientists. Thus, the standards committees were faced with the almost impossible task of modernising the language and making it safer to use, whilst at the same time keeping it 'Fortran-like' and efficient. Fortran 90 was the answer.

There were other justifications for continuing to revise the definition of the language. As well as standardizing vendor extensions, there was a need to respond to the developments in language design that had been exploited in other languages, such as APL, Algol 68, Pascal, Ada, C and C++. Here, X3J3 could draw on the obvious benefits of concepts like data hiding. In the same vein was the need to begin to provide an alternative to dangerous storage association, to abolish the rigidity of the outmoded source form, and to improve further on the regularity of the language, as well as to increase the safety of programming in the language and to tighten the conformance requirements. To preserve the vast investment in Fortran 77 codes, the whole of Fortran 77 was retained as a subset. However, unlike the previous standard, which resulted almost entirely from an effort to standardize *existing practices*, the Fortran 90 standard was much more a *development* of the language, introducing features that were new to Fortran, although based on experience in other languages. This tactic, in fact, proved to be highly controversial, both within the committee and with the wider community. Vested interests got in on the act, determined, depending on their persuasion, and in particular on whether they were users or vendors, either to extend Fortran to cope better with new computers and new problem domains or to stop the whole process in its tracks. The technical and political infighting reached legendary proportions. It was not until 1991, after much vigorous debate and thirteen years' work, that Fortran 90 was finally published by ISO.

It introduced a new notation that allows arrays of numbers, for instance matrices, to be handled in a natural and clear way, and added many new built-in

facilities for manipulating such arrays, for example, to add together all the numbers in an array, a single command (`sum`) is all that is required. The use of the array-handling facilities made scientific programming simpler, less error prone and, on the most powerful computers whose hardware can handle vectors of numbers, potentially more efficient than ever.

To make programs more reliable, the language introduced a wealth of features designed to catch programming errors during the early phase of compilation, when they can be quickly and cheaply corrected. These features included new ways of structuring programs and the ability to ensure that the components of a program, the subprograms, 'fit together' properly. For instance, Fortran 90 makes it simple to ensure that an argument mismatch can never arise as it enables programmers to construct verifiable interfaces between subprograms.

In summary, the main features of Fortran 90 were, first and foremost, the array language and data abstraction. The former is built on whole array operations and assignments, array sections, intrinsic procedures for arrays, and dynamic storage. It was designed with optimization in mind. The latter is built on modules and module procedures, derived data types, operator overloading and generic interfaces, together with pointers. Also important were the new facilities for numerical computation, including a set of numeric inquiry functions, the parameterization of the intrinsic types, new control constructs – `select case` and new forms of `do`, internal and recursive procedures and optional and keyword arguments, improved I/O facilities, and many new intrinsic procedures. Last but not least were the new free source form, an improved style of attribute-oriented specifications, the `implicit none` statement, and a mechanism for identifying redundant features for subsequent removal from the language. The requirement on compilers to be able to identify syntax extensions, and to report why a program had been rejected, was also significant. The resulting language was not only a far more powerful tool than its predecessor, but a safer and more reliable one too. Storage association, with its attendant dangers, was not abolished, but rendered unnecessary. Indeed, experience showed that compilers detected errors far more frequently than before, resulting in a faster development cycle. The array syntax and recursion also allowed quite compact code to be written, a further aid to safe programming. Fortran 90 also allowed programmers to tailor data types to their exact needs. Another advance was the

language's new ability to structure program data into arbitrarily complex patterns – lists, graphs, trees, etc. – and to manipulate these structures conveniently. This is achieved through the use of pointers. A related feature was the ability to allocate storage for program data dynamically.

After this revision, Fortran became, it must be admitted, a different language, as the entire issue of the journal [Computer Standards & Interfaces, Vol. 18 (1996)], which is devoted to various aspects of the development of Fortran 90, shows.

5. The Fifth Age: A minor revision, Fortran 95

Following the publication of Fortran 90, two further significant developments concerning the language occurred. The first was the continued operation of the two standards committees, J3 (as X3J3 became known) and the international WG5, and the second was the founding of the High Performance Fortran Forum (HPFF).

Early on in their deliberations, the committees decided on a strategy whereby a minor revision of Fortran 90 would be prepared by the mid-1990s and a further revision by about the year 2000. The first revision, Fortran 95, is the subject of this section.

The HPFF was set up in an effort to define a set of extensions to Fortran, such that it would be possible to write portable, single-threaded code when using parallel computers for handling problems involving large sets of data that can be represented by regular grids. This version of Fortran was to be known as High Performance Fortran (HPF), and Fortran 90 was chosen as the base language. Thus, HPF was a superset of Fortran 90, the main extensions being expressed in the form of directives. However, it did become necessary also to add some additional syntax, as not all of the desired features could be accommodated in the form of directives.

It was evident that, in order to avoid the development of divergent dialects of Fortran, it would be desirable to include the new syntax defined by HPF in Fortran 95 and, indeed, these features were the most significant new ones that Fortran 95 introduced. The other changes consisted mainly of what are known as corrections, clarifications and interpretations. Only a small number of other pressing but minor language changes were made.

A new ISO standard, replacing Fortran 90, was adopted in 1997.

6. The Sixth Age: Fortran 2003

Without a break, standardization continued, and the following language standard, Fortran 2003, was published, somewhat delayed, in 2004. The major enhancements were:

- Derived type enhancements: parameterized derived types, improved control of accessibility, improved structure constructors, and finalizers.
- Object-oriented programming support: type extension, inheritance, polymorphism, dynamic type allocation, and type-bound procedures.
- Data manipulation enhancements: deferred type parameters, the `volatile` attribute, explicit type specification in array constructors and allocate statements, pointer enhancements, extended initialization expressions, and enhanced intrinsic procedures.
- Input/output enhancements: asynchronous transfer, stream access, user-specified transfer operations for derived types, user-specified control of rounding during format conversions, named constants for pre-connected units, the flush statement, regularization of keywords, and access to error messages.
- Procedure pointers.
- Support for the exceptions of the IEEE Floating-Point Standard (IEEE 1989).
- Interoperability with the C programming language.
- Support for international usage: access to ISO 10646 4-byte characters and the choice of decimal or comma in numeric formatted input/output.
- Enhanced integration with the host operating system: access to command-line arguments, environment variables, and processor error messages.

In addition, there were numerous minor changes but Fortran 2003 was essentially upwards compatible with the Fortran 95 standard that it replaced. The enhancements had, after all, been developed in response to demands from users and to keep Fortran relevant to the needs of programmers, without losing the vast investment in existing programs.

Related standards

No Fortran standard up to and including Fortran 2003 included any significant feature intended directly to facilitate parallel programming. Rather, this has had

to be achieved through the intermediary of *ad hoc* industry standards, in particular HPF, MPI, OpenMP and Posix Threads.

HPF directives take the form of Fortran comment lines that are recognized as such only by an HPF processor. An example is

```
!HPF$ ALIGN WITH b :: a1, a2, a3
```

to align three conformable (matching in shape) arrays with a fourth, thus ensuring locality of reference. Further directives allow, for instance, aligned arrays to be distributed over a set of processors.

MPI is a library specification for message passing. OpenMP supports multiplatform, shared-memory parallel programming and consists of a set of compiler directives, library routines, and environment variables that determine runtime behaviour. Posix Threads is again a library specification, for multithreading.

MPI and OpenMP have both become widespread, but HPF has ultimately met with little success.

7. The Seventh Age: Fortran 2008

Notwithstanding the fact that Fortran 2003conformant compilers have been very slow to appear, the standardization committees proceeded with yet another standard, Fortran 2008. Its single most important new feature is coarray handling (described below). Further, the `do concurrent` form of loop control and the `contiguous` attribute are introduced. Other major new features include: sub-modules, enhanced access to data objects, enhancements to I/O and to execution control, and more intrinsic procedures, in particular for bit processing. Fortran 2008 was published in 2010 [3], and is the current standard.

Fortran concepts

Programming languages have many features in common. In this section some that represent Fortran’s special strengths are briefly outlined. Its object-oriented features are, however, omitted, but these and all the other features are fully described in [ISO/IEC 1539-1: 2010. ISO, Geneva, Switzerland. [4] Metcalf, M., Reid. J. and Cohen, M. (2011). *Modern Fortran Explained*. Oxford University Press, Oxford and New York.].

…

The status of Fortran

1. Challenges from other languages

Fortran has always had a slightly old-fashioned image. In the 1960s, the block-structured language Algol was regarded as superior to Fortran. In the 1970s the more powerful PL/1 was expected to replace Fortran. Algol's successors Pascal and Ada caused Fortran proponents some concern in the 1980s. Meanwhile, it continued successfully as the workhorse of scientific computing. However, by the late 1980s, two developments did begin seriously to impinge on Fortran's predominance in this field: Unix and object orientation.

Unix brought with it the highly-successful general-purpose language C, which was further developed into C++, an object-oriented language. C is widely used for all levels of system programming and made inroads into Fortran's traditional numerical computing community. C++ came to dominate many programming applications especially those requiring sophisticated program interfaces. Another object-oriented language, Java, has also come into widespread use.

Fortran's particular advantages as a high-end numerical language, especially where arrays are the main data object and/or where complex arithmetic is involved, remain. It is able to attain the highest achievable optimization, mainly because multidimensional arrays are 'natural' objects and because its pointers are highly constrained. Nevertheless, whether modern Fortran will, in the long term, be able to withstand the immense pressure from other languages remains an open question. However, there is every sign that Fortran continues to be used to tackle major scientific computing problems, and will long remain a living memorial to the early pioneers. Indeed, at a Workshop on Software in High-Energy Physics in 1982, I predicted that: *"Fortran is likely to remain into the next century as, at the very least, a special-purpose scientific and numerical language for large-scale, computing-intensive applications and, strengthened especially by its array capabilities, will be one of a small range of widely-used languages in general use"*. This turned out to be not too far from the truth!

2. The international Fortran community

Fortran is an international language both in the sense that it used throughout the world, and also in that the community of international users has, over the

last 30 years, actively participated in the development of the standards. Furthermore, the Internet and the World-Wide Web have facilitated the development of international user communities.

These groups are important in the dissemination of Fortran news, such as announcements of new compilers, and as sources of help and advice to users in general. The ACM publishes *Fortran Forum,* a special interest publication on Fortran with an international readership and containing articles on Fortran language developments and user experience.

We thus see that there is a healthy user community, even if the language now occupies, in contrast to the past, only a niche in the world of programming, but one nevertheless concerned with large and important applications. Long may it continue!

Bibliography: Fortran History

Michael Metcalf (2016, unpublished)

John Backus 1957 paper (republished)

Programming Systems and Languages S. Rosen, ed (McGraw Hill, 1967) pp. 29-47

First two textbooks (for Fortran II)

A Guide to Fortran Programming Daniel D McCracken (Wiley, 1961)

A Fortran Primer Elliott I Organick (Addison-Wesley, 1963)

Early history of Fortran (Fortran I to Fortran 77)

Programming Languages: History and Fundamentals Jean E Sammet (Prentice Hall, 1969)

The History of Fortran I, II, and III John Backus (ACM SIGPLan History of Programming Languages Conference - Preprints, published in ACM SIGPLan Notices 13:8, August 1978) pp. 165-180.

Programming language standardisation I D Hill and B L Meek, eds (Ellis Horwood, 1980) *Fortran,* W S Brainerd, Chapter 2, p. 34.

History of Programming Languages R L Wexelblat, ed (Academic Press, 1981) pp. 25-74.

Annals of the History of Computing (AFIP 6:1, January, 1984) whole issue

Encyclopedia of Science and Technology, vol. 5 (Academic Press, 1986) *Fortran*

Fortran 90

Fortran Optimization M Metcalf (Academic Press, 1982) Chapter 12, pp. 194-210. [A snapshot of the plans for Fortran 90 (then known as Fortran 8x), as foreseen in 1982. A later snapshot appears in the 1986 Edition.]

The Fortran (Not the Foresight) Saga: The Light and the Dark B Meek (SIGPLan Special Interest Publication on Fortran, Fortran Forum: ACM Press) Vol 9 No 2, Oct 1990 [just before Fortran 90 was adopted; also online at www.fortran.com/fortran/forsaga]

Fortran 90 Explained M Metcalf and J Reid (Oxford University Press, 1990) Chapter 1, pp. 3-8.

Encyclopedia of Science and Technology, vol. 6 (Academic Press, 1992) *Fortran* pp. 632-637.

Numerical Recipes in Fortran 90 M Metcalf (Cambridge University Press, 1996) Foreword, pp. x-xvi [Detailed.]

Computer Standards & Interfaces, 18 (North Holland/Elsevier, 1996) whole issue. [Articles devoted to Fortran 90 contain further background information.]

Imperative Programming Languages Handbook of Programming Languages, Vol II: Imperative Programming Languages P H Salus, ed (MacMillan, 1998) *Part I: Fortran*, W S Brainerd

Onward to Fortran 2008

Journal of Computer Science and Technology 11:1 M Metcalf *The Seven Ages of Fortran* (April 2011)

Encyclopedia of Parallel Computing D Padua, ed (Springer, 2011): *Fortran 90 and its successors*, M Metcalf

Modern Fortran Explained M Metcalf, J Reid and M Cohen (Oxford University Press, 2011) Chapter 1, pp. 4-7.