Annals of the History of Computing, Vol 1, No 1, Jan 1984
Special Issue: FORTRAN's Twenty-Fifth Anniversary
Pages 28-32
Scanned and transcribed by Loren Meissner, 6 May 2016

Institutionalization of FORTRAN

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.

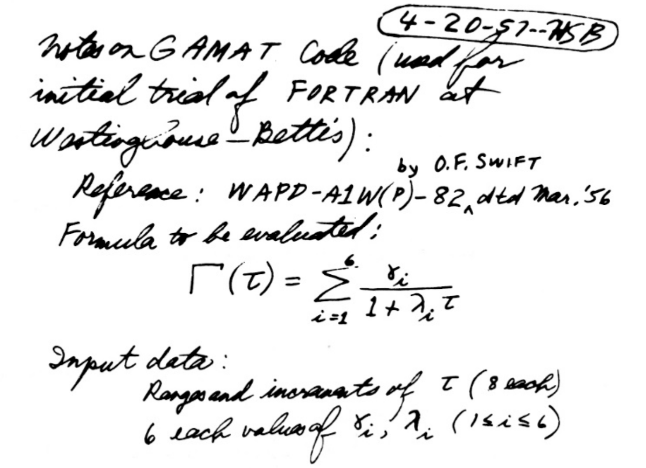
To simulate the reactor core through its working lifetime, it was necessary to perform a depletion or burnout calculation using the output of a criticality calculation to determine at each point inside the core what neutron bombardment had done to the core materials. That turned out to be an enormously 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 [Herb Bright, “FORTRAN Comes to Westinghouse-Bettis, 1957.” Annals of the History of Computing, 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.



4-20-57--HSB

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

Reference: WAPD-A1W(P)-82 by O.F. SWIFT dtd Mar. '56

Formula to be evaluated:

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

Input data:

Ranges and increments of τ (8 each)

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

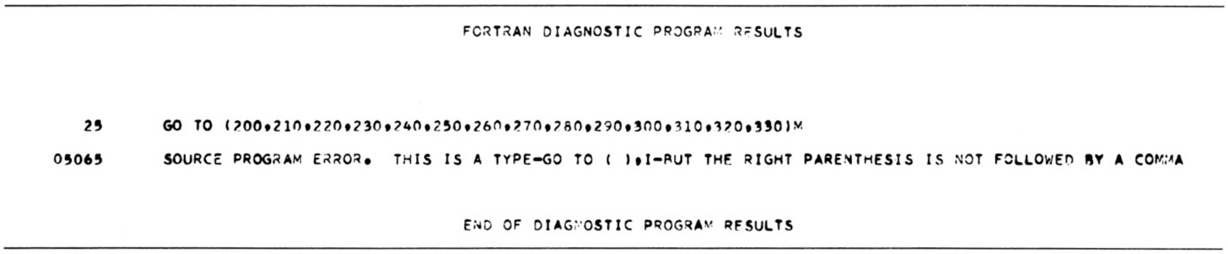
Figure 1. Notes on Gamma (Tau) code.

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.

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.”



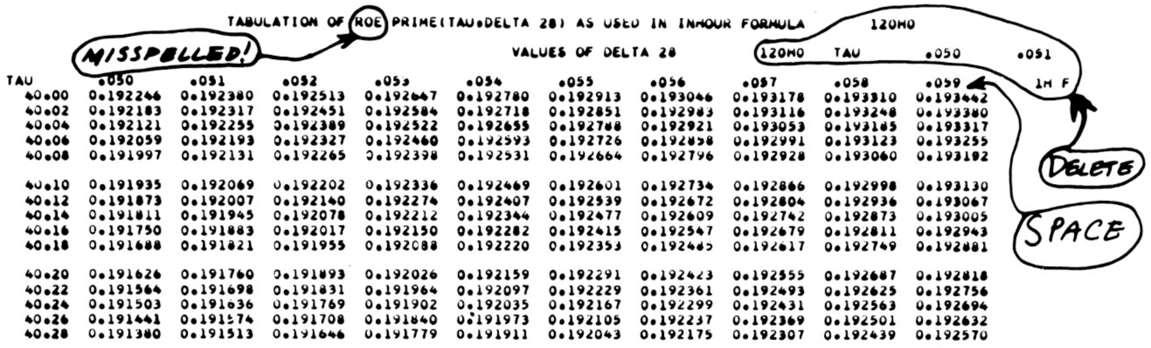
```
                              FORTRAN DIAGNOSTIC PROGRAM RESULTS


     25      GO TO (200,210,220,230,240,250,260,270,280,290,300,310,320,330)M
  05065      SOURCE PROGRAM ERROR.  THIS IS A TYPE-GO TO ( ),I-BUT THE RIGHT PARENTHESIS IS NOT FOLLOWED BY A COMMA

                              END OF DIAGNOSTIC PROGRAM RESULTS
```

Figure 2. Gamma (Tau) diagnostic printout.

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).



```
                    TABULATION OF ROE PRIME(TAU.DELTA 28) AS USED IN INHOUR FORMULA          120H0

                                                  VALUES OF DELTA 28               120H0     TAU       .050      .051

TAU         .050      .051      .052      .053      .054      .055      .056      .057      .058      .059       1H F
  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 jn 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).

Meetings in Retrospect

FORTRAN Activities at SHARE Meeting

Elliott C Nohr

Pages 65-69

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.