The History of FORTRAN I, II, and III
JOHN BADKUS

This article discusses attitudes about "automatic programming" systems with the exception of Lancing and Zierler's algebraic system [Lanning and Zierler 1954] and the A-2 compiler [Remington Rand 1953; Moser 1954]) simply provided a synthetic "computer" with an order code different from that of the real machine. This synthetic computer usually had floating point instructions and index registers and had improved input-output commands; it was therefore much easier to program than its real counterpart.

The A-2 compiler also came to be a synthetic computer sometime after early 1954. But in early 1954 it had a much clumsier form; instead of "pseudo-instructions" its input was then a complex sequence "compiling instructions" which could take a variety of forms ranging from machine code itself to lengthy groups of words constituting rather clumsy calling sequences for the desired floating point subroutine, "altered form-instructions" that were converted by a "Translator" into ordinary "compiling instructions" ([Moser 1954].

After May 1954 the A-2 compiler acquired a "pseudocode" which was similar to the order codes for many floating point interpretive systems that were already in operation in 1953; e.g., the Los Alamos systems, DUAL and SHACO [Bouricius 1953; Schlesinger 1953], the MIT "Summer Session Computer" [Adams and Lanning 1954], a system for the ILLIAC designed by D. J. Wheeler [Muller 1954], and the SPEEDCODING system for the IBM 701 (Buckx 1954). The Lanning and Zierler system was quite a different story; it was the world's first operating algebraic compiler, a rather elegant but simple one. Knuth and Pardo [1971] assign this honor to Alick Glennie's AUTOCODE, but I for one am unable to recognize the sample AUTOCODE program they give as "algebraic," especially when it is compared to the corresponding Lanning and Zierler program.

All of the early "automatic programming" systems were costly to use, since they slowed the machine down by a factor of five or ten. The most common reason for the slowdown was that these systems were spending most of their time in floating point subroutines. Simulated indexing and other "housekeeping" operations could be done with simple inefficient techniques, since, slow as they were, they took far less time than the floating point work.

Experience with slow "automatic program- ming" systems, plus their own experience with...
the problems of organizing loops and address modification efficiently to produce programs that were efficient was something that could not be automated. Another reason that "automatic programming" was not taken seriously by the computing community came from the energetic public relations efforts of some visionaries to spread the word that their "automatic programming" systems had almost human abilities to understand the language and needs of the user, whereas closer inspection of these same automatic programs revealed complex, exception-
edden performer of clerical tasks which was both difficult to use and inefficient. Whatever the reasons, it is obvious that it is not the fault of the later seventies the strength of the skepticism about "automatic programming." It is, rather, in general and about its ability to produce efficient programs in particular, as it existed in 1954.

The above discussion of attitudes about "automatic programming" in 1954 I have mentioned only those actual systems of which my colleagues and I were aware at the time. For a comprehensive treatment of early programming systems and languages I recommend the articles by Smith and Parlet [1977] and Sammet [1960] et al.

1.2 The economics of programming

Another factor which influenced the development of FORTRAN was the economics of programming in 1954. The cost of programmers associated with a computer was usually at least as much as the cost of the computer itself. (This fact follows from the average salary-plus-benefits of typical programmers at each center and from the computer rental figures.) In addition, from one quarter to one half of the computer's time was spent in debugging. Thus programming and debugging accounted for as much as three quarters of the cost of operating a computer, and, obviously, as computers got cheaper, this situation would get worse. The high cost was one of the prime motivations which led me to propose the FORTRAN project in a letter to my boss, Guthrie, in late 1953. Guthrie, in a letter of January 1954, provided for our constantly expanding needs over the next five years without ever asking us to project or justify those needs in a formal budget. The viability of most compilers and interpreters prior to FORTRAN and ALGOL is that it was difficult for most source language operations were not machine

1.3 Programming systems in 1974

The programmer of today to comprehend what "automatic programming" meant to programmers in 1954. To many it then meant simply providing monomeric operation codes and symbolic addresses, to others it meant the simple process of obtaining subroutines from a library and inserting the addresses of operands and result into each subroutine. Most of the automatic-programming systems were either assembly languages or subroutine-filling programs, or, most popularly, the ones which provided floating point and indexing operations. My friends and I were aware of a number of assembly programs and subroutine-filling programs which have already been mentioned above; besides these there were about two others: we were already considering A-2 compiler [Remington Rand 1955, Moyer 1954] and the Lanning and Zierler [1954] algebraic compiler as MIT. As noted above, the A-2 compiler was at that time largely a subroutine-filling [is its original task was to provide for "overlays"]; but from the standpoint of input the program provided fewer conveniences than most of the then current interpretive systems mentioned earlier; it later adopted a "pseudo-code" as input which was similar to the input codes of these interpretive systems. The Lanning and Zierler system accepted as input an elegant but rapidly machine algebraic language. It permitted single-letter variables (identifiers) and a set of integer variables. The repertoire of functions one could use were denoted by "F" with an associated argument list for each number of the desired function. Algebraic expressions were compiled into closed subroutines and placed on a magnetic drum for subsequent use. The system was originally designed for the Whirlwind III and had 1,054 storage cells with the result that it caused a slowdown in execution speed by a factor of about twenty. The other two systems were attempts at a common goal, in my mind would be widely used only if we could demonstrate that it would produce programs almost as efficient as hand-coded ones and do so virtually every job.

The viability of most compilers and interpreters prior to FORTRAN and ALGOL is that it was difficult for most source language operations were not machine

1954 letter [Backus 1954-a] (by which Dr. Lanning knew me) to write FORTRAN for a demonstration of his system. It makes clear that we had learned of his work at the Office of Naval Research Symposium on Automatic Programming for Digital Computers, May 13-14, 1954, and that the demonstrated performance took place on June 2, 1954. The letter also makes clear that the FORTRAN project was well under way when the later one was invoked. Our colleagues Harlan Herrick, Robert A. Nelson, and Irving Zierler were as well as myself. Furthermore, an article in the proceedings of that same ONR Symposium by Herrick and myself [Backus and Herrick 1954] described the input of input expressions like "\( Z_{ab} \), and "\( X \times Y \)." We went on to raise the question "...can a machine translate a sufficiently rich mathematical language into a sufficiently economical program at a sufficiently low cost to make the whole affair feasible?"

These and other remarks in our paper presented at the Symposium in May 1954 make it clear that we were already considering algebraic input considerably more sophisticated than that of Lanning and Zierler's system when we first heard of their pioneering work. Though Lanning and Zierler project had already produced the world's first algebraic compiler, our basic idea for FOR- TRAN was to develop a compiler independent; thus it was difficult to know what, if any, new ideas we got from seeing the demonstration of their system.

Our ONR Symposium article [Backus and Herrick 1954] also makes clear that the FOR- TRAN group was already aware that it faced a new kind of problem in automatic programming.

1954 (June 23, 1954)] as discussed in Naur and Perlis [1977]. We were aware of but not influenced by the automatic programming efforts which came before such systems as the SAGE "SAGE Computing System," SHAC, DIAL, SPEEDCODING, and the ILLIAC system, since these were considerably different from those of FORTRAN. Nor were we influenced by algebraic systems which were designed after our "Preliminary Report" [1954] but which began operation after FORTRAN (e.g., SISYAC, Green and Partner 1966). We were aware of the work of Church and Zadeh [1956], which was our own work but which also presented a form of symbolic language implementation. However, the use of FORTRAN was by far the most advanced and the most widespread of those that appeared ten and fifteen years later.

The FORTRAN project was now a part of the Computing Laboratory of the University of Utah, but the work of this project was largely done at IMRL, a component of the ONR, and was funded by the ONR.
It was our belief that if FORTRAN, during its
first months, were to translate any reasonable
"eventive" sequence of program into an object
program only half as fast as its hand coded counter-
part, then acceptance of our system would be in
serious danger. This belief caused us to regard
the design of the translator as the real challenge,
not the simple task of designing the language.
Our belief in the simplicity of language design
was based on the relative ease with which
similar languages had been independently
developed by Rutishauser [1952], Lining and
Zieter [1954], and ourselves; whereas we were
designed alone in seeking to produce really efficient object
programs.

This is, I believe that our emphasis on
object program efficiency rather than on lan-
guage design was basically correct. I believe that
had we failed to produce efficient programs, the widespread use of languages like FORTRAN
would have been seriously delayed. In fact, I believe that we are a similar, but unrecognized,
situation today; in spite of all the facts that has been made over myriad language details, current
conventional languages are still very weak pro-
gramming aids, and far more powerful languages
would be in use today if anyone had found a way
to make them run with adequate efficiency. In
other words, the next revolution in programming
will take place only when both of the following
requirements have been met: (a) a new kind of
programming language, far more powerful than
those we have developed and (b) a technique
has been found for executing its programs at
no much greater cost than that of today's.

Because of our 1954 view that success in
producing efficient programs was more impor-
tant than the design of the FORTRAN language,
I consider the history of the compiler construc-
tors and of FORTRAN an integral part of the
history of the FORTRAN language; therefore a later section deals with that subject.

2. The Early Stages of the FORTRAN
Program

After Cuthbert Hurd approved my proposal to
develop a practical automatic programming system
on December 15 or January 10, 1954, Ivan Ziller
was assigned to the project. We started work on
one of the many tasks involved in the vicinity of
IBM headquarters at 590 Madison Avenue in
New York; the first of these was in the Jay Thorpe
Building on Fifth Avenue. By May 1954 we had
an operation by a new
employee who had been hired to do technical
typing; Robert A. Nelson (with Ziller, he later
began designing one of the most sophisticated
sections of the compiler; he is now an IBM
Fellow). By about May we had moved to the 10th
floor of the same building, next to the elevator
machinery; the ground floor of the building
was reserved for the famous IBM elevators, where
which customers tested their programs before
the arrival of their own machines. It was there
that the bulk of the FORTRAN system was
written, mostly by Herrick, Ziller and myself, except
that some sections were done by others. The
organization that we were designing for IBM
was then, and for many years, called Roy Nutt,
employee of United Aircraft Corp., who was soon to become a
member of the FORTRAN project. After we had
finished designing most of the language we heard about Rota's proposal for a similar
language [Rutishauser 1952]. It was characteristic of the
scholarly attitude of most programmers
then, and of ourselves in particular, that we did
not bother to carefully review the sketchy transla-
tion of his proposals that we finally obtained,
since from their symbolic content they did not
appear to add anything new to our proposed
language. Rutishauser's language had a for-
statement and one-dimensional arrays, but no IF,
GOTO, nor IF 1 statement. Subscript variables
could not be used as ordinary variables and
operator precedence was ignored. In 1952
we had proposed a paper "Preliminary Report,
Specifications for the IBM Mathematical FORmu-
la TRANslating System, FORTRAN" (Prelimi-
nary Report 1954) dated November 10. In its
introduction we noted that "systems which have
ought to reduce the job of coding and debug-
ging programs have offered the choice of easy
coding and slow execution or laborious coding
and fast execution." On the basis more of faith
than of knowledge, we suggested that programs
"will be executed in about the same time that
would be required had the program been labori-
ously hand coded." In what turned out to be a
true statement, we said that "FORTRAN may
apply complex, lengthy techniques in coding a
problem which the human coder would have
required more than one generation to derive or
demonstrate to an automatic program." The
language described in the "Preliminary Report"
had variables of one or two characters in
length, function names of three or more charac-
ters, recursively defined "expressions," subscripted
variables with up to three subscript,
"arithmetic formulas" (which turn out to be assigned
to variables), and "DO-formulas": after writing
these latter formulas could specify the first and
last indices to list other variables to be controlled, thus permitting
a single sentence to control a large
sequence of statements, as well as specifying a third
statement to which control would pass following the end
of the iteration. If only one statement was specified,
the "range" of the DO was the sequence of
statements following the DO down to the spec-
cified statement.

We felt that these provided a good basis for achieving
good code for a variety of floating point
problems. A "floating point" quantity could be
mixed either "fixed point" (integer) and
"floating point" quantities. The arithmetic used
in FORTRAN (all integer or all floating point) is evaluated
by a mixed expression was determined by the
mixing rules set in the definition of the
"arithmetic formulas" employed an equality or inequality
sign ("=" or "=" or ">"), but the second
was restricted), each followed by the statement
numbers, one for the "false" case, the other for
the "true" case.

A "Relabel formula" was designed to make it
easy to repeat, so the rows of the for statement, it is permitted to change the
labels, but only after relabeling, even though a new
row had been read in and the next computation was
not to take place until all labels had been read.

The "next input statements" provided included
the basic notion of specifying the sequence in
which data was to be read in or out, but did not include any "format" statements.

The Report also lists four kinds of "spe-
cification sentences": (1) "dimension specifications" for
giving the dimensions of arrays, (2) "equivalence sentences" for assigning the same
storage locations to variables, (3) "frequen-
cy sentences" for indicating estimated relative frequency of branch paths or loops to help the
compiler optimize the object program, and
(4) "relative constant sentences" to indicate sub-
scripts and the like. This was not well done,
and we are not at all sure how their values were
computed.

The final section of the Report (pp. 28-29)
discusses programming techniques to help
the system produce efficient programs. It dis-

II

I discuss one had another task to believe (since which merits few were whose real sium vom 1979 problem has facilities some Relative scripted The I, "Since section 1956]

and some (thus some term not be checking facilities the page formu-\[!]."

sup-

15, in later counting of input-output

DO

of our

using

to FORTRAN Report. (see Los

of our language and we had worked out the basic sort of machine programs which we wanted the compiler to generate for various source language phrases; Ziller and I had worked out a basic scheme for doing so.

But the real work on the compiler got under way in our third location on the 15th floor of 15 East 56th Street. By the middle of February three separate efforts were under way. The first two of these concerned, respectively, the compiler and the third concerned the input, output and assembly programs we were going to need (see below). We combined these efforts, and the third effort would produce most of the compiler.

(All the compiler project was carried on using a loose natural mathematical language that appeared there in the paper by Herrick and myself [Backus and Herrick 1954]. But it was our impression that our discussions with various groups after that time, their access to our Preliminary Report, and their awareness of the extent and seriousness of our efforts, that these factors either gave the initial stimulus to some other projects or at least caused them to be more likely to amount to something. It was our impression, for example, that the "TT" project [Perlin, Purdue and later at Carnegie-Mellon began shortly after the distribution of our Preliminary Report [Backus et al. 1957] at Spero Rand].

It is our experience, if any, our Los Angeles talk and earlier contacts with members of their group had on the PACT I effort [Baker 1956], which I believe was already in for some stages when we got to Los Angeles. It is clear, whatever influence the specifications for FORTRAN may have had on other projects in 1954—55—56, that our plans were well advanced and quite firm by the end of 1954 and before we had contact or knowledge of those other projects. Our specifications were not affected by them in any significant way as far as I am aware, even though some were being operated before FORTRAN was (since they were primarily intended in providing an input language rather than in the implementation, their task was considerably simpler than ours).

3. The Construction of the Compiler

The FORTRAN compiler (or "translator" as we called it then) was begun in early 1955, although a lot of work went into it before that used in it had been done in 1954; e.g., Herrick had done a lot of trial programming to test our language and we had worked out the basic sort of machine programs which we wanted the compiler to generate for various source language phrases; Ziller and I had worked out a basic scheme for doing so.

But the real work on the compiler got under way in our third location on the 15th floor of 15 East 56th Street. By the middle of February three separate efforts were under way. The first two of these concerned, respectively, the compiler and the third concerned the input, output and assembly programs we were going to need (see below). We combined these efforts, and the third effort would produce most of the compiler.

(All the compiler project was carried on using a loose
cooperation between autonomous, separate groups of people; each group was responsible for a "section" of the compiler; each group gradually developed and agreed upon its own input and output specifications with the groups for neighboring sections; each group invented and programmed the necessary techniques for doing its assigned job.

Section 1 was to read the entire source program from cards, punch cards, or tape, and record all the rest of the information from the source program on cards; this the compiler was "reading". Thus the compiler was "one pass" in the sense that it "saw" the source program only once. After reading, the compilers were being clocked by DO statements. Despite this simplicity, the assumption, the number of cases that section 1 had to analyze was enormous; it had to analyze optimal assignment of near-optimal code was very large. The number of sections that section 1 was selecting with this number of subsets; this was a prime factor in our decision to limit them to three; the fact that the 704 had only three index registers was a

It is beyond the scope of this paper to go into the details of the analysis which section 2 carried out. It will suffice to say that it produced code of such efficiency that its output would startle the programmers who studied it. It moved code out of loops where that was possible; it took advantage of the differences between row-wise and column-wise scans; it took note of special cases to optimize even the exits from loops. The degree of optimization performed by section 2 in its treatment of indexing, array references, and other cases not equalized on the methods used in late compilers began to appear in the middle and late sixties.

The architecture and all the techniques employed in section 2 were invented by Robert A. Nelson and Irving Ziller. They planned and programmed the entire section. Originally it was intended to be another section, and the address of (L1, I, and K) was chosen for their area, including the choice of the index register to be used (the 704 had three index registers). When they started working on it, the prophetic idea blossomed into the realization that it rapidly became clear that it was not going to be easy to treat it optimally. At that point 1 proposed that they should produce a program for a 704 with an unlimited number of index registers, and they went on to analyze the frequency of execution of various parts of the program (by a Monte Carlo simulation) and set up what would be called register assignments so as to minimize the transfers in the internal and the index registers. This proposal gave rise to two new sections of the compiler which we had not anticipated, sections 3 and 4.

The first section of the compiler, section 6, assembled all the information into a single relocatable binary program (it could also be used as a relocatable binary program in SPC, the SHARE Assembly Program for the 704). It is in the nature of the program and data that was a complete summary of the FORTRAN output. Of course it also contained the necessary inclusion register usage for more details see [Bucca et al. 1957; Cooke and Schwartz 1970, p. 511]. Section 3 would then do the actual transformation of the program from one having an unlimited number of index registers to one having only three again. Thus, the section entirely planned and programmed by Halib.

Section 5 was planned and programmed by Sheldon Best, who were loaned to our group by his employer, Charles W. Adams, at the Digital Computer Laboratory at MIT; during his stay with us Best was a temporary IBM employee. Starting in the early fall of 1955, he designated what turned out to be, along with section 2, one of the most intricate and complex sections of the compiler, one which had perhaps more impact on the methods used in later compilers than any other part of the FORTRAN compiler. For a discussion of his techniques see [Cooke and Schwartz 1970 pp. 510-515]. It is impossible to describe his register allocation method here; it suffices to say that it was to become the basis for much subsequent work and produced code very difficult to improve.

Another problem believe that no provably optimal register allocation algorithm is known for general programs with loops, etc., empirically Best's 1953-56 procedure appeared to be optimal. For straight-line code Best's replacement policy was the same as used in Bellady's MCM algorithm, which Bellady proved to be optimal [Bellady 1965]. Although Best did not publish a formal proof, he came up with arguments for the optimality of his policy in the early 1955. It was recognized that yet another section, section 5, was needed. This section merged the outputs of the preceding sections into a single uniform 704 program which could refer to any number of index registers. It was planned and programmed by Richard Goldberg, a mathematician who joined our group in November 1955. Alto, also in 1956, after Best had returned to MIT and during the debugging of the system, section 5 was taken over by Goldberg and David Sayre (see below), who diagrammed it carefully and took charge of the debugging. The final section of the compiler, section 6, assembled all the information into a single relocatable binary program (it could also be used as a relocatable binary program in SPC, the SHARE Assembly Program for the 704). It is in the nature of the program and data that was a complete summary of the FORTRAN output. Of course it also contained the necessary inclusion register usage for more details see [Bucca et al. 1957; Cooke and Schwartz 1970, p. 511].
been optimized beyond recognition, and Goldberg or Sayre could tell us how section 5 had generated additional indexing operations). Transfers of control appeared which corresponded to no source statement, expressions were radically rewritten, and the same DO statement might produce no instructions in the object program in one context, and in another it would produce many instructions in different places in the program.

By the summer of 1955 what appeared to be the immense completion of the project started to worry (for perhaps the first time) about documentation. David Sayre, a crystallographer who had joined us in the spring the had earlier consulted with Best on the design of section 5 and had later begun serving as second-in-command of what was now the "Programming Research Department" took up the task of writing the Programmer's Reference Manual [IBM 1956]. It appeared in a glossy cover, handsomely printed, with the date October 15, 1956. It stood for some time as a unique example of a manual for a programming language (perhaps it still does): it had wide margins, yet was only 31 pages long. Its description of the FORTRAN language, exclusive of input-output statements, was 21 pages, the I/O description occupied another 11 pages: the rest of it was examples and details about arithmetic, tables, etc., it gave an elegant recursive definition of expressions (as given by Searle), and concise, clear descriptions, with examples, of each statement type, of which there were 32, mostly machine dependent items like SENSE LIGHT, IF DIVE CHECK, PUNCH, READ DRUM, and so on. (For examples of its style see Figs. 1, 2, and 3.1)

One feature of FORTRAN I is missing from the Programmer's Reference Manual, not from an oversight of Sayre's, but because it was added to the system after the manual was distributed. This feature was the ability to define a function by a "function statement." These statements had to precede the rest of the program. They looked like assignment statements, with the defined function and dummy arguments on the left and an expression involving those arguments on the right. They are described in the addenda to the programmer's Reference Manual [Addenda 1957] which we sent on February 8, 1957 to John Gremstal, who was in charge of IBM's facility for distributing information to SHARE. They are also described in all subsequent material on FORTRAN I.

The next documentation task we set ourselves was to write a paper describing the FORTRAN language and the translator program. The result was a paper entitled "The FORTRAN automatic coding system" [Backus et al. 1957] which we presented at the Western Joint Computer Conference in Los Angeles in February 1957. I have mentioned all of the thirteen authors of that paper in the preceding narrative except one: Robert A. Hughes. He was employed by the Livermore Radiation Laboratory, by arrangement with Sidney Fernbach, he visited us for two or three months in the summer of 1956 to help us document the system. (The authors of that paper were: J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, L. M. Halib. H. L. Herrick, R. A. Hughes, R. A. Nelson, R. Nust, D. Sayre, P. B. Sheridan, H. Stern, J. Ziller.)

At about the time of the Western Joint Computer Conference we spent some time in Los Angeles still frantically debugging the system. North American Aviation gave us time at night on their 704 to help us in our mad rush to distribute the system. Up to this point there had been relatively little interest from 704 installations (with the exception of Rancho's United Aircraft shop, Harry Cottrell's GE installation in Schenectady, and Sidney Fernbach's Livermore operations), but now that the full system was beginning to generate object programs interest picked up in a number of places.

Sometime in early April 1957 we felt the system was sufficiently bug-free to distribute to all 704 installations. Sayre and Grace Mitchell (see below) started to punch out the binary decks of the system, each of about 2,000 cards, with the intention to make 30 or 40 decks for distribution. This process was so error-prone that they had to give up after spending an entire night in producing only one or two decks.

(Apparently one of those decks was sent, without any identification or directions, to the Westinghouse-Betz installation, where a puzzled group headed by Herbert S. Bright, suspecting that it might be the long-awaited FORTRAN deck, proceeded, entirely by guesswork, to get it to compile a test program—a clever diagnostic printout noting that a comma was missing in a specific statement! This program then printed 28 pages of correct results—with a few FORMAT errors. The date: April 20, 1957. An amusing...
Any such routine will be compiled into the object program as a closed subroutine. In the section on Writing Subroutines for the Master Tape in Chapter 7 are given the specifications which any such routine must meet.

Expressions

An expression is any sequence of constants, variables (subscripted or not subscripted), and functions, separated by operation symbols, commas, and parentheses so as to form a meaningful mathematical expression.

However, one special restriction does exist. A Fortran expression may be either a fixed or a floating point expression, but it must not be a mixed expression. This does not mean that a floating point quantity cannot appear in a fixed point expression, or vice versa, but rather that a quantity of one mode can appear in an expression of the other mode only in certain ways. Briefly, a floating point quantity can appear in a fixed point expression only as an argument of a function; a fixed point quantity can appear in a floating point expression only as an argument of a function, or as a subscript, or as an exponent.

Formal Rules for Forming Expressions: By repeated use of the following rules, all permissible expressions may be derived.

1. Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same mode. Thus 5 and 'A' are fixed point expressions, and ALPHA and A(J,K) are floating point expressions.

2. If SOMEF is some function of n variables, and if E, F, G, H, ... are a set of expressions of the correct modes for SOMEF, then SOMEF(E, F, G, H, ...) is an expression of the same mode as SOMEF.

3. If E is an expression, and if its first character is not = or +, then E and E+ are expressions of the same mode as E. Thus 'A' is an expression, but +A is not.

4. If E is an expression, then (E) is an expression of the same mode as E. Thus (A), (A+), (I+1), etc. are expressions.

5. If E and F are expressions of the same mode, and if the first character of F is not + or =, then E+F, E-F, E*F, E/F, E+F, E-F, E*F, E/F are expressions of the same mode. Thus +A-B and A/-B are not expressions. The characters +, -, *, /, and . denote addition, subtraction, multiplication, and division.

Account of this incident by Bright in Comput. and Autom. (Bright 1971).

After failing to produce binary decks, Sayre devised and programmed the simple editor and loader that made it possible to distribute and update the system from magnetic tapes, this arrangement served as the mechanism for creating new system tapes from a master tape and the binary correction cards which our group would generate in large numbers during the long field debugging and maintenance period which followed distribution.

With the distribution of the system tapes went a "Preliminary Operator's Manual" [Operator's Manual 1957] dated April 8, 1957. It describes how to use the tape editor and how to maintain the library of functions. Five pages of such general instructions are followed by 52 pages of error stops, many of these say "source program error, get off machine, correct formula in question and restart problem" and then, for example (stop 3024) "non-zero level reduction due to insufficient or redundant parentheses in arithmetical or IF-else formula." Shortly after the distribution of the system we distributed—some copy per installation—what was fondly known as the "Tome," the complete symbolic listing of the entire compiler plus other system and diagnostic information, an "11" by "15" volume about four or five inches thick.

The proprietors of the six sections were kept busy tracking down bugs elicited by our customers' use of FORTRAN until the late summer of 1957. Hal Serrin served as the coordinator of the field debugging and maintenance efforts; he received a stream of telegrams, mail and phone calls from all over the country and distributed the incoming problems to the appropriate members of our group to track down the errors and generate correction cards, which he then distributed to every installation.

In the spring of 1957 Grace E. Mitchell joined our group to write the Programmer's Manual [1] for FORTRAN. The Primer was divided into three sections: each described successively larger subsets of the language accompanied by many example programs. The first edition of the Primer was issued in the late fall or winter of 1957; a slightly revised edition appeared in March 1958. Mitchell planned and wrote the 64-page Primer with some consultation with the rest of the group; the latest programme most of the extensive changes in the system which resulted in FORTRAN II (see below).

The Primer had an important influence on the subsequent growth in the use of the system. I believe it was the only available simplified instruction manual (other than reference manuals) until the later appearance of books such as McCracken 1961, (Organick 1963) and many others.

A report on FORTRAN usage in November 1958 (Buckus 1958) says that in April 1958 of twenty-six 704 installations indicates that over half of them use FORTRAN I for more than half of their problems. Many use it for 80% or more of their work...and almost all use it for some of their work." By the fall of 1958 there were some 60 installations with about 66 704's and...more than half the machine instructions for these machines are being produced by FORTRAN. SHARE recently designated FORTRAN as the second official medium for transmittal of programs (SAP was the first)...
STOP

<table>
<thead>
<tr>
<th>GENERAL FORM</th>
<th>EXAMPLES</th>
</tr>
</thead>
</table>
| "STOP" or "STOP n" where n is an unsigned octal line point constant | STOP
| STOP 7777 |

This statement causes the machine to HALT in such a way that pressing the START button has no effect. Therefore, in contrast to the PAUSE, it is used where a get-off-the-machine stop, rather than a temporary stop, is desired. The octal number is displayed on the 704 console in the address field of the storage register. (If n is not stated it is taken to be 0.)

**DO**

<table>
<thead>
<tr>
<th>GENERAL FORM</th>
<th>EXAMPLES</th>
</tr>
</thead>
</table>
| 'DO i = m, n or 'DO i = m, n, ni where m is a statement number, i is a non-subscripted fixed point variable, and nj are each either an unsigned fixed point constant or a non-subscripted fixed point variable. | 00 301 = 1, 10
| 00 301 = 1, n, 3 |

The DO statement is a command to "DO the statements which follow, to and including the statement with statement number n", repeatedly, the first time with i = m, and with i increased by m for each succeeding time; after they have been done with i equal to the highest of this sequence of values which does not exceed m, let control reach the statement following the statement with statement number n". The range of a DO is the set of statements which will be executed repeatedly, it is the sequence of consecutive statements immediately following the DO, to and including the statement numbered n.

The index of a DO is the fixed point variable i, which is controlled by the DO in such a way that its value begins at m, and is increased each time by m until it is about to exceed m, Throughout the range it is available for computation, either as an ordinary fixed point variable or as the variable of a subscript. During the fast execution of the range, the DO is said to be satisfied. For example, that control has reached statement 10 of the program 10 DO I = 1, 10
11 A(I) = I
12 STOP

FORTRAN II (with the most unusual feature that she delivered it ahead of schedule). She was aided in this by Berlyne Brady and LeRoy May. Sheridan planned and made the necessary changes in his part of section I; Nast did the same for section III. FORTRAN II was distributed in the spring of 1958.

5. FORTRAN III

While FORTRAN II was being developed, Ziller was designing even more advanced system that he called FORTRAN III. It allowed one to write intermixed symbolic instructions and FORTRAN statements. The symbolic (704) instructions could have FORTRAN variables (with or without subscript(s)) as "addresses." In addition to this machine dependent feature (which assured the demise of FORTRAN III along with that of the 704), it contained early versions of a number of improvements that were later to appear in FORTRAN IV. It had "Boolean" expressions, function and subroutine names could be passed as arguments, and it had facilities for handling alphanumeric data, including a new FORMAT code "A" similar to codes "I" and "E." This system was planned and programmed by Ziller with some help from Nelson and Nast. Ziller maintained it and made it available to about 20 (mostly IBM) installations. It was never distributed generally. It was accompanied by a brief descriptive document [Additions to FORTRAN II 1958]. It became available on this limited scale in the winter of 1958-59 and was in operation until the early sixties, in part on the 709 using the compatibility feature (which made the 709 order code the same as that of the 704).

6. FORTRAN After 1958: Comments

By the end of 1958 or early 1959 the FORTRAN group (the Programming Research Department), while still helping with an occasional debugging problem with FORTRAN II, was primarily occupied with other research. Another IBM department had long since taken responsibility for the FORTRAN system and was revising it in the course of producing a translator for the 709 which used the same procedures at the 704 FORTRAN II translator. Since my friends and I no longer had responsibility for FORTRAN and were busy thinking about other things by the end of 1958, that seems like a good point to break off this account. There remains only a few comments to be made about the subsequent development of FORTRAN.

The most obvious defect in FORTRAN II for many of its users was the time spent in compiling. Even though the facilities of FORTRAN II permitted separate compilation of subroutines and hence eliminated the need to recompile an entire program at each step in debugging it, nevertheless compile times were long and, during debugging, the considerable time spent in optimizing was wasted. I repeatedly suggested to those who were in charge of FORTRAN that they should now develop a fast compiler and/or interpreter without any optimizing at all for use during debugging and for short-run jobs. Unfortunately the developers of FORTRAN IV thought they could have the best of both worlds in a single compiler, one which was both fast and produced optimized code. I was unsuccessful in convincing them that two compilers would have been far better than the compromise which became the original FORTRAN IV compiler. The latter was not nearly as fast as later compilers like WATFOR (Crest, Dirksen and Graham 1970) nor did it produce as good code as FORTRAN II. (For more discussion of later developments with FORTRAN see Backus and Heising [1961].)

My own opinion as to the effect of FORTRAN on later languages and the collective impact of such languages on programming generally is not a popular opinion. That viewpoint is the subject of a long paper [Backus 1978]. I now regard all conventional languages (e.g., FORTRAN, ALGOL, FORTRAN IV, etc.) as increasingly complex elaborations of the style of programming dictated by von Neumann's computer. These "von Neumann languages" create enormous, unnecessary intellectual roadblocks in thinking about programs and in creating the higher level combining forms required in a really powerful programming methodology. Von Neumann languages constantly keep our noses pressed in the dirt of address computation and the separate computation of single words, whereas we should be focusing on the form and content of the overall result we are trying to produce. We have come to regard the DO, FOR, WHILE statements and the like as powerful tools, whereas they are in fact weak palliatives that are necessary to make the primitive von Neumann style of programming viable at all.

By splitting programming into a world of expressions on the one hand and a world of statements on the other, von Neumann languages prevent the effective use of higher level combining forms; the lack of the latter makes the
definitional capabilities of von Neumann languages so weak that most of their important features cannot be defined—starting with a small, elegant framework—but must be built into the framework of the language at the outer. The Gorgonzola size of recent von Neumann languages is eloquent proof of their inability to define new constructs: for no one would build in so wild encodings and collusions if they could be defined and fit into the existing framework later.

The world of expressions has some elegant and useful mathematical properties whereas the structure of expressions is a disorderly one, without useful mathematical properties. Structured programming can be viewed as a modest effort to impose some order into the chaotic world of statements. The work of Hoare [1969], Dijkstra [1972] and others to axiomatize the properties of the statement world can be viewed as a valiant and effective effort to probe about those properties, ungauged as they may be.

It is not the place for me to elaborate any further any views about von Neumann languages. My point is this: while it was perhaps natural and inevitable that languages like FORTRAN and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. It is necessary because their (disorderly) familiar-}

ity will make it hard for us to understand and use unstructured programming techniques which one day will offer far greater intellectual and computational power.

Acknowledgements

My greatest debt in connection with this paper is to all those individuals and colleagues whose creative work, hard work and invention made FORTRAN possible. While I cannot acknowledge them all, I wish to acknowledge their gratitude to them for their contributions to the project, for making our work together so long and easy a congenial and memorable experience, and, more recently, for providing me with a great amount of help and support in the preparation of this paper and for their valuable efforts of an easier draft. For all this I thank all those individuals and colleagues who were NOTyon language and who, though not so numerous, are listed here. In particular, I must thank those who were the principal movers in making FORTRAN a reality. Sheldon Best, Richard Goldberg, Lois Haibt, Harlan Herrick, Grace Mitchell, Robert Nelson, Roy Roy, David Sayre, Peter Sheridan, and Irving Zeller.

I also wish to thank Bernard Galler, J.A. Lee, and Henry Tropp for their amiable, extensive and invaluable suggestions for improving the first draft of this paper. I am grateful too for all the work of the Program Committee in preparing FORTRAN I and for all the suggestions from editors and the anonymous reviewers that suggested a number of topics in the paper.

REFERENCES

Most of the items listed below have dates in the fifties, they are so applegate that there will be unobtainable only in the largest and oldest collections. The items with an asterisk were either not published or were not translated into English.


Addenda to the FORTRAN Programmer's Reference Manual. 1957 February 8. (Transmitted to Dr. John Greenough, Special Program Group, Applied Science Division, IBM, for distribution to SHARE members, by letter from John W. Backus, Programming Research Dept. IBM, 5 pages.)

Additions to FORTRAN I. 1958. Description of Strong Language additions to the FORTRAN II System. New York: Programming Research, IBM Corp. (Distributed to SHARE members, 20 pp.)


Baker, Charles L. 1956 October. The PACT I coding system for the IBM Type 701. JACM (4):272-278.


