Comm., from A.V. Astin, Chief, Electronics and Ordance Div., Nov. 18, 1949, "Status of Work on Automatic Electronic Computing Machine" (National Archives).

[60]See Correspondence, John W. Mauchly to E. Mauchly, Vice President, Woolworth Corp., Jan. 24, 1948, which outlines the types of contracts EMCC was hoping to obtain. (Moore School Archives).

[61]"Projects and Publications of the National Applied Mathematics Laboratories," NBS, June 1948, p. 3.

[62]See J. H. Curtiss, "A Federal Program in Applied Mathematics," *Science 107* , Mar. 12, 1948; "History of the National Bureau of Standards Program for the Development and Construction of Large-Scale Electronic Computing Machines," undated NBS publication.

[63]EMCC, Interoffice Memo, Aug. 18, 1948 (Moore School Archives).

[64]EMCC, Interoffice Memo, Aug. 28, 1948 (Moore School Archives).

[65]EMCC, Interoffice Memo, John Mauchly to Executive Committee, Aug. 18, 1948: "Ike Auerbach has, during the past few months, maintained that we could produce additional BINAC computers within six months from the time at which the Northrop BINAC was delivered." (Moore School Archives).

Scanned

The History of FORTRAN I, II, and III

JOHN BACKUS

This article discusses attitudes about "automatic programming," the economics of programming, and existing programming systems, all in the early 1950s. It describes the formation of the FORTRAN group, its knowledge of existing systems, its plans for FORTRAN, and the development of the language in 1954. It describes the development of the optimizing compiler for FORTRAN I, of various language manuals, and of FORTRAN II and III. It concludes with remarks about later developments and the impact of FORTRAN and its successors on programming today.

Key words and phrases: programming language, history, compilers, optimization, language design, compiler design, FORTRAN, programming language manuals

CR categories: 1.2, 4.2

1. Early Background and Environment

1.1 *Attitudes about automatic programming in the 1950s*

Before 1954 almost all programming was done in machine language or assembly language. Programmers rightly regarded their work as a complex, creative art that required human inventiveness to produce an efficient program. Much of their effort was devoted to overcoming the difficulties created by the computers of that era: the lack of index registers, the lack of built-in floating point operations, restricted instruction sets (which might have AND but not OR, for example), and primitive input-output arrangements. Given the nature of computers, the services which "automatic programming" performed for the programmer were concerned with overcoming the machine's shortcomings. Thus the primary concern of some "automatic programming" systems was to allow the use of symbolic addresses and decimal numbers (e.g., the MIDAC Input Translation PROGRAM [Brown and Carr 1954]).

But most of the larger "automatic programming" systems (with the exception of Laning and Zierler's algebraic system [Laning 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 its input had a much cruder form; instead of "pseudo-instructions" its input was then a complex sequence of "compiling instructions" that 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, to "abbreviated 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 Laning 1954], a system for the ILLIAC designed by D.J. Wheeler [Muller 1954], and the SPEEDCODING system for the IBM 701 [Backus 1954].

The Laning 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 [1977] 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 Laning 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 programming" systems, plus their own experience with

Author's address: IBM Research Laboratory, San Jose, CA 95193.

the problems of organizing loops and address modification, had convinced programmers that efficient programming 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 systems would often reveal a complex, exception-ridden performer of clerical tasks which was both difficult to use and inefficient. Whatever the reasons, it is difhfult to convey to a reader in the late seventies the strength of the skepticism about "automatic programming" in general and about its ability to produce efficient programs in particular, as it existed in 1954.

(In 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 Knuth and Pardo [1977] and Sammet [1969].)

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 center was usually at least as great as the cost of the computer itself. (This fact follows from the average salary-plus-overhead and number of 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.

This economic factor was one of the prime motivations which led me to propose the FORTRAN project in a letter to my boss, Cuthbert Hurd, in late 1953 (the exact date is not known but other facts suggest December 1953 as a likely date). I believe that the economic need for a system like FORTRAN was one reason why IBM and my successive bosses, Hurd, Charles DeCarlo, and John McPherson, 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.

1.3 *Programming systems in 1954*

It is difficult for a programmer of today to comprehend what "automatic programming" meant to programmers in 1954. To many it then meant simply providing mnemonic operation codes and symbolic addresses, to others it meant the simple process of obtaining subroutines from a library and inserting the addresses of operands into each subroutine. Most "automatic programming" systems were either assembly programs, or subroutine-fixing programs, or, most popularly, interpretive systems to provide floating point and indexing operations. My friends and I were aware of a number of assembly programs and interpretive systems, some of which have been mentioned above; besides these there were primarily two other systems of significance: the A-2 compiler [Remington Rand 1953; Moser 1954] and the Laning and Zierler [1954] algebraic compiler at MIT. As noted above, the A-2 compiler was at that time largely a subroutine-fixer (its other principal task was to provide for "overlays"); but from the standpoint of its input "programs" it provided fewer conveniences than most of the then current interpretive systems mentioned earlier; it later adoped a "pseudo-code" as input which was similar to the input codes of these interpretive systems.

The Laning and Zierler system accepted as input an elegant but rather simple algebraic language. It permitted single-letter variables (identifiers) which could have a single constant or variable subscript. The repertoire of functions one could use were denoted by "F" with an integer superscript to indicate the "catalog 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 computer when it had 1,024 storage cells with the result that it caused a slowdown in execution speed by a factor of about ten [Adams and Laning 1954].

The effect of the Laning and Zierler system on the development of FORTRAN is a question which has been muddled by many misstatements on my part. For many years I believed that we had gotten the idea for using algebraic notation in FORTRAN from seeing a demonstration of the Laning and Zierler system at MIT. In preparing a paper [Backus 1976] for the International Research Conference on the History of Computing at Los Alamos (June 10-15 1976), I reviewed the matter with Irving Ziller and obtained a copy of a

1954 letter [Backus 1954a] (which Dr. Laning kindly sent to me). As a result the facts of the matter have become clear. The letter in question is one I sent to Dr. Laning asking 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 demonstration took place on June 2, 1954. The letter also makes clear that the FORTRAN project was well under way when the letter was sent (May 21, 1954) and included Harlan Herrick, Robert A. Nelson, and Irving Ziller as well as myself. Furthermore, an article in the proceedings of that same ONR Symposium by Herrick and myself [Backus and Herrick 1954] shows clearly that we were already considering input expressions like "$\Sigma a_{ij} \cdot b_{jk}$" and "X + 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 Laning and Zierler's system when we first heard of their pioneering work. Thus although Laning and Zierler had already produced the world's first algebraic compiler, our basic ideas for FORTRAN had been developed independently; thus it is difficult to know what, if any, new ideas we got from seeing the demonstration of their system.[1]

Our ONR Symposium article [Backus and Herrick 1954] also makes clear that the FORTRAN group was already aware that it faced a new kind of problem in automatic programming. The viability of most compilers and interpreters prior to FORTRAN had rested on the fact that most source language operations were not machine operations. Thus even large inefficiencies in performing both looping/testing operations and computing addresses were masked by most operating time being spent in floating point subroutines. But the advent of the 704 with built-in floating point and indexing radically altered the situation. The 704 presented a double challenge to those who wanted to simplify programming; first it removed the raison d'être of earlier systems by providing in hardware the operations they existed to provide; second, it increased the problem of generating efficient programs by an order of magnitude by speeding up floating point operations by a factor of ten and thereby leaving inefficiencies nowhere to hide. In view of the widespread skepticism about the possibility of producing efficient programs with an automatic programming system and the fact that inefficiencies could no longer be hidden, we were convinced that the kind of system we had in 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 on virtually every job.

John Backus led the development of FORTRAN, participated in designing the IBM 704 and ALGOL; he proposed "BNF" for syntax description. Currently at the IBM Research Laboratory, San Jose, California, he is developing a functional style of programming and its associated algebra. He received the National Medal of Science in 1975 and the Turing Award (ACM) in 1977.

[1] In response to suggestions of the Program Committee let me try to deal explicitly with the question of what work might have influenced our early ideas for FORTRAN, although it is mostly a matter of listing work of which we were then unaware. I have already discussed the work of Laning and Zierler and the A-2 compiler. The work of Heinz Rutishauser [1952] is discussed later on. Like most of the world (except perhaps Rutishauser and Corrado Böhm—who was the first to describe a compiler in its own language [Böhm 1954]) we were entirely unaware of the work of Konrad Zuse [1959; 1972]. Zuse's "Plankalkül," which he completed in 1945, was, in some ways, a more elegant and advanced programming language than those that appeared ten and fifteen years later.

We were also unaware of the work of Mauchly et al. ("Short Code," 1950), Burks ("Intermediate PL," 1950), Böhm (1951), Glennie ("AUTOCODE," 1952) as discussed in Knuth and Pardo [1977]. We were aware of but not influenced by the automatic programming efforts which simulated a synthetic computer (e.g., MIT "Summer Session Computer," SHACO, DUAL, SPEEDCODING, and the ILLIAC system), since their languages and systems were so different from those of FORTRAN. Nor were we influenced by algebraic systems which were designed after our "Preliminary Report [1954] but which began operation before FORTRAN (e.g., BACAIC [Grems and Porter 1956], IT [Perlis, Smith and Van Zoeren 1957], MATH-MATIC [Ash et al. 1957]). Although PACT I [Baker 1956] was not an algebraic compiler, it deserves mention as a significant development, designed after the FORTRAN language but in operation before FORTRAN; it did not influence our work.

It was our belief that if FORTRAN, during its first months, were to translate any reasonable "scientific" source program into an object program only half as fast as its hand coded counterpart, 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 partly confirmed by the relative ease with which similar languages had been independently developed by Rutishauser [1952], Laning and Zierler [1954], and ourselves; whereas we were alone in seeking to produce really efficient object programs.

To this day I believe that our emphasis on object program efficiency rather than on language 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 in a similar, but unrecognized, situation today: in spite of all the fuss that has been made over myriad language details, current conventional languages are still very weak programming 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 of today, has been developed and (b) a technique has been found for executing its programs at not much greater cost than that of today's programs.

Because of our 1954 view that success in producing efficient programs was more important than the design of the FORTRAN language, I consider the history of the compiler construction and the work of its inventors 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 Project

After Cuthbert Hurd approved my proposal to develop a practical automatic programming system for the 704 in December 1953 or January 1954, Irving Ziller was assigned to the project. We started work in one of the many small offices the project was to occupy 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 been joined by Harlan Herrick and then by a new employee who had been hired to do technical typing. Robert A. Nelson (with Ziller, he soon 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 19th floor of the annex of 590 Madison Avenue, next to the elevator machinery; the ground floor of this building housed the 701 installation on which customers tested their programs before the arrival of their own machines. It was here that most of the FORTRAN language was designed, mostly by Herrick, Ziller and myself, except that most of the input-output language and facilities were designed by Roy Nutt, an 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 Rutishauser's proposals for a similar language [Rutishauser 1952]. It was characteristic of the unscholarly attitude of most programmers then, and of ourselves in particular, that we did not bother to carefully review the sketchy translation 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 I/O statements. Subscript variables could not be used as ordinary variables and operator precedence was ignored. His 1952 article described two compilers for this language (for more details see [Knuth and Pardo 1977]).

As far as we were aware, we simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which could produce efficient programs. Of course one of our goals was to design a language which would make it possible for engineers and scientists to write programs themselves for the 704. We also wanted to eliminate a lot of the bookkeeping and detailed, repetitive planning which hand coding involved. Very early in our work we had in mind the notions of assignment statements, subscripted variables, and the DO statement (which I believe was proposed by Herrick). We felt that these provided a good basis for achieving our goals for the language, and whatever else was needed emerged as we tried to build a way of programming on these basic ideas.

We certainly had no idea that languages almost

identical to the one we were working on would be used for more than one IBM computer, not to mention those of other manufacturers. (After all, there were very few computers around then.) But we did expect our system to have a big impact, in the sense that it would make programming for the 704 very much faster, cheaper, more reliable. We also expected that, if we were successful in meeting our goals, other groups and manufacturers would follow our example in reducing the cost of programming by providing similar systems with different but similar languages [Preliminary Report 1954].

By the fall of 1954 we had become the "Programming Research Group" and I had become its "manager." By November of that year we had produced a paper: "Preliminary Report, Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN" [Preliminary Report 1954] dated November 10. In its introduction we noted that "systems which have sought to reduce the job of coding and debugging problems 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 problem been laboriously 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 neither the time nor inclination to derive or apply."

The language described in the "Preliminary Report" had variables of one or two characters in length, function names of three or more characters, recursively defined "expressions," subscripted variables with up to three subscripts, "arithmetic formulas" (which turn out to be assignment statements), and "DO-formulas." These latter formulas could specify both the first and last statements to be controlled, thus permitting a DO to control a distant 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 specified statement.

Expressions in "arithmetic formulas" could be "mixed": involve both "fixed point" (integer) and "floating point" quantities. The arithmetic used (all integer or all floating point) to evaluate a mixed expression was determined by the type of the variable on the left of the "=" sign. "IF-formulas" employed an equality or inequality sign ("=" or ">" or ">=") between two (restricted) expressions, followed by two statement numbers, one for the "true" case, the other for the "false" case.

A "Relabel formula" was designed to make it easy to rotate, say, the indices of the rows of a matrix so that the same computation would apply, after relabelling, even though a new row had been read in and the next computation was now to take place on a different, rotated set of rows. Thus, for example, if b is a 4 by 4 matrix, after RELABEL (3, 1), a reference to b(1,j) has the same meaning as b(3,j) before relabelling; b(2,j) after = b(4,j) before; b(3,j) after = b(1,j) before; and b(4,j) after = b(2,j) before relabelling.

The input-output 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 "specification sentences": (1) "dimension sentences" for giving the dimensions of arrays, (2)"equivalence sentences" for assigning the same storage locations to variables, (3) "frequency 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 subscript variables which are expected to change their values very infrequently.

Toward the end of the Report (pp. 26-27) there is a section "Future additions to the FORTRAN system." Its first item is: "a variety of new input-output formulas which would enable the programmer to specify various formats for cards, printing, input tapes and output tapes." It is believed that this item is a result of our early consultations with Roy Nutt. This section goes on to list other proposed facilities to be added: complex and double precision arithmetic, matrix arithmetic, sorting, solving simultaneous equations, differential equations, and linear programming problems. It also describes function definition capabilities similar to those which later appeared in FORTRAN II; facilities for numerical integration; a summation operator; and table lookup facilities.

The final section of the Report (pp. 28-29) discusses programming techniques to use to help the system produce efficient programs. It discusses how to use parentheses to help the system identify identical subexpressions within an expression and thereby eliminate their duplicate calculation. These parentheses had to be supplied only when a recurring subexpression occurred as part of a term (e.g., if a*h occurred in several places, it would be better to write the term a*b*c as (a*b)*c to avoid duplicate calculation); otherwise the system would identify duplicates without any assistance. It also observes that the system would not produce optimal code for loops constructed without DO statements.

This final section of the Report also notes that "no special provisions have been included in the FORTRAN system for locating errors in formulas." It suggests checking a program "by independently recreating the specifications for a problem from its FORTRAN formulation [!]." It says nothing about the system catching syntactic errors, but notes that an error-finding program can be written after some experience with errors has been accumulated.

Unfortunately we were hopelessly optimistic in 1954 about the prohlems of debugging FORTRAN programs (thus we find on page 2 of the Report: "Since FORTRAN should virtually eliminate coding and debugging...[!]") and hence syntactic error checking facilities in the first distribution of FORTRAN I were weak. Better facilities were added not long after distribution and fairly good syntactic checking was provided in FORTRAN II.

The FORTRAN language described in the *Programmer's Reference Manual* dated October 15, 1956 [IBM 1956] differed in a few respects from that of the Preliminary Report, but, considering our ignorance in 1954 of the problems we would later encounter in producing the compiler, there were remarkably few deletions (the Relabel and Relative Constant statements), a few retreats, some fortunate, some not (simplification of DO statements, dropping inequalities from IF statements—for lack of a ">" symbol, and prohibiting most "mixed" expressions and subscripted subscripts), and the rectification of a few omissions (addition of FORMAT, CONTINUE, computed and assigned GOTO statements, increasing the length of variables to up to six characters, and general improvements of input-output statements).

Since our entire attitude ahout language design had always been a very casual one, the changes which we felt to be desirahle during the course of writing the compiler were made equally casually. We never felt that any of them involved a real sacrifice in convenience or power (with the possible exception of the Relabel statement, whose purpose was to coordinate input-output with computations on arrays, but this was one facility which we felt would heen really difficult to implement). I believe the simplification of the original DO statement resulted from the realization that (a) it would be hard to describe precisely, (b) it was awkward to compile, and (c) it provided little power beyond that of the final version.

In our naive unawareness of language design problems—of course we knew nothing of many issues which were later thought to be important, e.g., block structure, conditional expressions, type declarations—it seemed to us that once one had the notions of the assignment statement, the subscripted variable, and the DO statement in hand (and these were among our earliest ideas), then the remaining problems of language design were trivial: either their solution was thrust upon one by the need to provide some machine facility such as reading input, or by some programming task which could not be done with existing structures (e.g., skipping to the end of a DO loop without skipping the indexing istructions there: this gave rise to the CONTINUE statement).

One much-criticized design choice in FORTRAN concerns the use of spaces: blanks were ignored, even blanks in the middle of an identifier. Roy Nutt reminds me that that choice was partly in recognition of a problem widely known in SHARE, the 704 users' association. There was a common problem with keypunchers not recognizing or properly counting blanks in handwritten data, and this caused many errors. We also regarded ignoring blanks as a device to enable programmers to arrange their programs in a more readahle form without altering their meaning or introducing complex rules for formatting statements.

Another debatable design choice was to rule out "mixed" mode expressions involving both integer and floating point quantities. Although our Preliminary Report had included such expressions, and rules for evaluating them, we felt that if code for type conversion were to be generated, the user should be aware of that, and the best way to insure that he was aware was to ask him to specify them. I believe we were also doubtful of the usefulness of the rules in our Report for evaluating mixed expressions. In any case, the most common sort of "mixture" was

allowed: integer exponents and function arguments were allowed in a floating point expression.

In late 1954 and early 1955, after completing the Preliminary Report, Harlan Herrick, Irving Ziller and I gave perhaps five or six talks about our plans for FORTRAN to various groups of IBM customers who had ordered a 704 (the 704 had been announced about May 1954). At these talks we covered the material in the Report and discussed our plans for the compiler (which was to be completed within about six months [!]; this was to remain the interval-to-completion until it actually was completed over two years later, in April 1957). In addition to informing customers about our plans, another purpose of these talks was to assemble a list of their objections and further requirements. In this we were disappointed; our listeners were mostly skeptical; I believe they had heard too many glowing descriptions of what turned out to be clumsy systems to take us seriously. In those days one was accustomed to finding lots of peculiar but significant restrictions in a system when it finally arrived that had not been mentioned in its original description. Most of all, our claims that we would produce efficient object programs were disbelieved. Whatever the reasons, we received almost no suggestions or feedback; our listeners had done almost no thinking about the problems we faced and had almost no suggestions or criticisms. Thus we felt that our trips to Washington (D.C.), Albuquerque, Pittsburgh, Los Angeles, and one or two other places were not very helpful.

One trip to give our talk, probably in January 1955, had an excellent payoff. This talk, at United Aircraft Corp., resulted in an agreement between our group and Walter Ramshaw at United Aircraft that Roy Nutt would become a regular part of our effort (although remaining an employee of United Aircraft) to contribute his expertise on input-output and assembly routines. With a few breaks due to his involvement in writing various SHARE programs, he would thenceforth come to New York two or three times a week until early 1957.

It is difficult to assess the influence the early work of the FORTRAN group had on other projects. Certainly the discussion of Laning and Zierler's algebraic compiler at the ONR Symposium in May 1954 would have been more likely to persuade someone to undertake a similar line of effort than would the brief discussion of the merits of using "a fairly 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 effors, that these factors either gave the initial stimulus to some other projects or at least caused them to be more active than they might have been otherwise. It was our impression, for example, that the "IT" project [Perlis, Smith and Von Zoeren 1957] at Purdue and later at Carnegie-Mellon began shortly after the distribution of our Preliminary Report, as did the "MATH-MATIC" project [Ash et al. 1957] at Sperry Rand.

It is not clear what influence, 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 its formative 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 operating before FORTRAN was (since they were primarily interested in providing an input language rather than in optimization, their task was considerable 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 on various schemes which would be used in it had been done in 1954; e.g., Herrick had done a lot of trial programming to test out 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 translating arithmetic expressions.

But the real work on the compiler got under way in our third location on the fifth floor of 15 East 56th Street. By the middle of February three separate efforts were under way. The first two of these concerned sections 1 and 2 of the compiler, and the third concerned the input, output and assembly programs we were going to need (see below). We believed then that these efforts would produce most of the compiler.

(The entire project was carried on by a loose cooperation between autonomous, separate groups of one, two, or three 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, compile what instructions it could, and file all the rest of the information from the source program in appropriate tables. Thus the compiler was "one pass" in the sense that it "saw" the source program only once. Herrick was responsible for creating most of the tables, Peter Sheridan (who had recently joined us) compiled all the arithmetic expressions, and Roy Nutt compiled and/or filed the I/O statements. Herrick, Sheridan and Nutt got some help later on from R.J. Beeber and H. Stern, but they were the architects of section 1 and wrote most of its code. Sheridan devised and implemented a number of optimizing transformations on expressions [Sheridan 1959] which sometimes radically altered them (of course without changing their meaning). Nutt transformed the I/O "lists of quantities" into nests of DO statements which were then treated by the regular mechanisms of the compiler. The rest of the I/O information he filed for later treatment in section 6, the assembler section. (For further details about how the various sections of the compiler worked see [Backus et al. 1957].)

Using the information that was filed in section 1, section 2 faced a completely new kind of problem: it was required to analyze the entire structure of the program in order to generate optimal code from DO statements and references to subscripted variables. The simplest way to effect a reference to A(I,J) is to evaluate an expression involving the address of A(1,1), I, and K×J, where K is the length of a column (when A is stored column-wise). But this calculation, with its multiplication, is much less efficient than the way most hand coded programs effect a reference to A(I,J), namely, by adding an appropriate constant to the address of the preceding reference to the array A whenever I and J are changing linearly. To employ this far more efficient method section 2 had to determine when the surrounding program was changing I and J linearly.

Thus one problem was that of distinguishing between, on the one hand, references to an array element which the translator might treat by incrementing the address used for a previous reference, and those array references, on the other hand, which would require an address calculation starting from scratch with the current values of the subscripts.

It was decided that it was not practical to track down and identify linear changes in subscripts resulting from assignment statements. Thus the sole criterion for linear changes, and hence for efficient handling of array references, was to be that the subscripts involved were being controlled by DO statements. Despite this simplifying assumption, the number of cases that section 2 had to analyze in order to produce optimal or near-optimal code was very large. (The number of such cases increased exponentially with the number of subscripts; this was a prime factor in our decision to limit them to three; the fact that the 704 had only three index registers was not a factor.)

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 loops was not equalled again until optimizing compilers began to appear in the middle and late sixties.

The architecture and all the techinques employed in section 2 were invented by Robert A. Nelson and Irving Ziller. They planned and programmed the entire section. Originally it was their intention to produce the complete code for their area, including the choice of the index register to be used (the 704 had three index registers). When they started looking at the problem it rapidly became clear that it was not going to be easy to treat it optimally. At that point I proposed that they should produce a program for a 704 with an unlimited number of index registers, and that later sections would analyze the frequency of execution of various parts of the program (by a Monte Carlo simulation of its execution) and then make index register assignments so as to minimize the transfers of items between the store and the index registers.

This proposal gave rise to two new sections of the compiler which we had not anticipated, sections 4 and 5 (section 3 was added still later to convert the output of sections 1 and 2 to the form required for sections 4, 5, and 6). In the fall of 1955 Lois Mitchell Haibt joined our group to plan and program section 4, which was to analyze the flow of a program produced by sections 1 and 2, divide it into "basic blocks" (which contained no branching), do a Monte Carlo (statistical) analysis of the expected frequency of execution of basic blocks—by simulating the behavior of the program and keeping counts of the use of each block—using information from DO statements, and FREQUENCY statements, and collect information about index register usage (for more details see [Backus et al. 1957; Cocke and Schwartz 1970, p. 511]). Section 5 would then do the actual transformation of the program from one having an unlimited number of index registers to one having only three. Again, the section was entirely planned and programmed by Haibt.

Section 5 was planned and programmed by Sheldon Best, who was loaned to our group by agreement with 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 designed 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 influence on the methods used in later compilers than any other part of the FORTRAN compiler. (For a discussion of his techniques see [Cocke 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.

Although I believe that no provably optimal register allocation algorithm is known for general programs with loops, etc., empirically Best's 1955–56 procedure appeared to be optimal. For straight-line code Best's replacement policy was the same as that used in Belady's MIN algorithm, which Belady proved to be optimal [Belady 1965]. Although Best did not publish a formal proof, he had convincing arguments for the optimality of his policy in 1955.

Late in 1955 it was recognized that yet another section, section 3, 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 us in November 1955. Also, late 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 its final debugging.

The final section of the compiler, section 6, assembled the final program into a relocatable binary program (it could also produce a symbolic program in SAP, the SHARE Assembly Program for the 704). It produced a storage map of the program and data that was a compact summary of the FORTRAN output. Of course it also obtained the necessary library programs for inclusion in the object program, including those required to interpret FORMAT statements and perform input-output operations. Taking advantage of the special features of the programs it assemhled, this assembler was about ten times faster than SAP. It was designed and programmed by Roy Nutt, who also wrote all the I/O programs and the relocating binary loader for loading object programs.

By the summer of 1956 large parts of the system were working. Sections 1, 2, and 3 could produce workable code provided no more than three index registers were needed. A number of test programs were compiled and run at this time. Nutt took part of the system to United Aircraft (sections 1, 2, and 3 and the part of section 6 which produced SAP output). This part of the system was productive there from the summer of 1956 until the complete system was available in early 1957.

From late spring of 1956 to early 1957 the pace of debugging was intense; often we would rent rooms in the Langdon Hotel (which disappeared long ago) on 56th Street, sleep there a little during the day and then stay up all night to get as much use of the computer (in the headquarters annex on 57th Street) as possible.

It was an exciting period; when later on we began to get fragments of compiled programs out of the system, we were often astonished at the surprising transformations in the indexing operations and in the arrangement of the computation which the compiler made, changes which made the object program efficient but which we would not have thought to make as programmers ourselves (even though, of course, Nelson or Ziller could figure out how the indexing worked, Sheridan could explain how an expression had 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 rearranged, 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 1956 what appeared to be the imminent completion of the project started us worrying (for perhaps the first time) about documentation. David Sayre, a crystallographer who had joined us in the spring (he 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 51 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 Sheridan), and concise, clear descriptions, with examples, of each statement type, of which there were 32, mostly machine dependent items like SENSE LIGHT, IF DIVIDE CHECK, PUNCH, READ DRUM, and so on. (For examples of its style see Figs. 1, 2, and 3.)

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 Greenstadt, 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. Haibt, H. L. Herrick, R. A. Hughes, R. A. Nelson, R. Nutt, D. Sayre, P. B. Sheridan, H. Stern, I. 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 Ramshaw's United Aircraft shop, Harry Cantrell's GE installation in Schenectady, and Sidney Fernbach's Livermore operation), 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-Bettis 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—after a 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

Figs. 1, 2, 3. Original FORTRAN Manual pages

Subscripts.

GENERAL FORM		EXAMPLES
Let v represent any fixed point variable		I
and c (or c') any unsigned fixed point		3
constant. Then a subscript is		MU+2
an expression of one of the forms:	v	MU−2
	c	5 * J
	v+c or v−c	5 * J+2
	c * v	5 * J−2
	c * v+c' or c * v−c'	

The symbol * denotes multiplication. The variable v must not itself be subscripted.

Subscripted Variables.

GENERAL FORM	EXAMPLES
A fixed or floating point variable	A(I)
followed by parentheses enclosing 1, 2, or 3	K(3)
subscripts separated by commas.	BETA(5 * J−2, K+2,L)

For each variable that appears in subscripted form the size of the array (i.e. the maximum values which its subscripts can attain) must be stated in a DIMENSION statement (see Chapter 6) preceding the first appearance of the variable.

The minimum value which a subscript may assume in the object program is +1.

Arrangement of Arrays in Storage.

A 2-dimensional array A will, in the object program, be stored sequentially in the order $A_{1,1}, A_{2,1}, \ldots\ldots, A_{m,1}, A_{1,2}, A_{2,2}, \ldots\ldots, A_{m,2}, \ldots\ldots\ldots, A_{m,n}$. Thus it is stored "columnwise", with the first of its subscripts varying most rapidly, and the last varying least rapidly. The same is true of 3-dimensional arrays. 1-dimensional arrays are of course simply stored sequentially. All arrays are stored backwards in storage; i.e. the above sequence is in the order of decreasing absolute location.

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 can not 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 3 and I are fixed point expressions, and ALPHA and A(I,J,K) are floating point expressions.
2. If SOMEF is some function of n variables, and if E, F, ... , H are a set of n expressions of the correct modes for SOMEF, then SOMEF (E, F, ... , 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)), (((A))), 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

   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 is in *Computers and Automation* [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 32 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 3624) "non-zero level reduction due to insufficient or redundant parentheses in arithmetic or IF-type formula." Shortly after the distribution of the system we distributed—one 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 Stern served as the coordinator of the field debugging and maintenance effort; 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 Primer* [1957] 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; she later programmed 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 [Backus 1958] says that "a survey 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] ..."

4. FORTRAN II

During the field debugging period some shortcomings of the system design, which we had been aware of earlier but had no time to deal with, were constantly coming to our attention. In the early fall of 1957 we started to plan ways of correcting these shortcomings; a document dated September 25, 1957 [Proposed Specifications 1957] characterizes them as (a) a need for better diagnostics, clearer comments about the nature of source program errors, and (b) the need for subroutine definition capabilities. (Although one FORTRAN I diagnostic would pinpoint, in a printout, a missing comma in a particular statement, others could be very cryptic.) This document is titled "Proposed Specifications for FORTRAN II for the 704"; it sketches a more general diagnostic system and describes the new "subroutine definition" and END statements, plus some others which were not implemented. It describes how symbolic information is retained in the relocatable binary form of a subroutine so that the "binary symbolic subroutine [BSS] loader" can implement references to separately compiled subroutines. It describes new prologues for these subroutines and points out that mixtures of FORTRAN-coded and assembly-coded relocatable binary programs could be loaded and run together. It does not discuss the FUNCTION statement that was also available in FORTRAN II. FORTRAN II was designed mostly by Nelson, Ziller, and myself. Mitchell programmed the majority of new code for

STOP

GENERAL FORM	EXAMPLES
"STOP" or "STOP n" where n is an unsigned octal fixed point constant.	STOP STOP 77777

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 n 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

GENERAL FORM	EXAMPLES
"DO n i = m_1, m_2" or "DO n i = m_1, m_2, m_3" where n is a statement number, i is a non-subscripted fixed point variable, and m_1, m_2, m_3 are each either an unsigned fixed point constant or a non-subscripted fixed point variable. If m_3 is not stated it is taken to be 1.	DO 30 I = 1, 10 00 30 I = 1, M, 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_1 and with i increased by m_3 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_2 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_1 and is increased each time by m_3 until it is about to exceed m_2. Throughout the range it is available for computation, either as an ordinary fixed point variable or as the variable of a subscript. During the last execution of the range, the DO is said to be *satisfied*.

Suppose, for example, that control has reached statement 10 of the program

```
10   DO 11 I = 1, 10
11   A(I) = I*N(I)
12
```


FORTRAN II (with the most unusual feature that she delivered it ahead of schedule). She was aided in this by Bernyce Brady and LeRoy May. Sheridan planned and made the necessary changes in his part of section 1; Nutt did the same for section 6. FORTRAN II was distributed in the spring of 1958.

5. FORTRAN III

While FORTRAN II was being developed, Ziller was designing an 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 subscripts) 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 Nutt. 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 as 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 remain 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 [Cress, 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 1964].)

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., the FORTRANs, the ALGOLs, their successors and derivatives) as increasingly complex elaborations of the style of programming dictated by the von Neumann 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 outset. The Gargantuan size of recent von Neumann languages is eloquent proof of their inability to define new constructs: for no one would build in so many complex features if they could be defined and would fit into the existing framework later on.

The world of expressions has some elegant and useful mathematical properties whereas the world of statements is a disorderly one, without useful mathematical properties. Structured programming can be viewed as a modest effort to introduce a small amount of order into the chaotic world of statements. The work of Hoare [1969], Dijkstra [1976] and others to axiomatize the properties of the statement world can be viewed as a valiant and effective effort to be precise about those properties, ungainly as they may be.

This is not the place for me to elaborate any further my 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 unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computational power.

Acknowledgments

My greatest debt in connection with this paper is to my old friends and colleagues whose creativity, hard work and invention made FORTRAN possible. It is a pleasure to acknowledge my gratitude to them for their contributions to the project, for making our work together so long ago such a congenial and memorable experience, and, more recently, for providing me with a great amount of information and helpful material in preparing this paper and for their careful reviews of an earlier draft. For all this I thank all those who were associated with the FORTRAN project but who are too numerous to list here. In particular I want to 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 Nutt, David Sayre, Peter Sheridan, and Irving Ziller.

I also wish to thank Bernard Galler, J.A.N. 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 helpful questions that suggested a number of topics in the paper.

REFERENCES

Most of the items listed below have dates in the fifties, thus many that appeared in the open literature will be obtainable only in the largest and oldest collections. The items with an asterisk were either not published or were of such a nature as to make their availability even less likely than that of the other items.

Adams, Charles W. and Laning, J.H., Jr. 1954 May. The MIT systems of automatic coding: Comprehensive, Summer Session, and Algebraic. In *Proc. Symp. on Automatic Programming for Digital Computers.* Washington DC: The Office of Naval Research.

**Addenda to the FORTRAN Programmer's Reference Manual.* 1957 February 8. (Transmitted to Dr. John Greenstadt, Special Programs 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 II. 1958. *Description of Source Language Additions to the FORTRAN II System.* New York: Programming Research, IBM Corp. (Distributed to users of FORTRAN III. 12 pages.)

*Ash, R.; Broadwin, E.; Della Valle, V.; Katz, C.; Green, M.; Jenny, A.; and Yu, L. 1957. *Preliminary Manual for MATH-MATIC and ARITH-MATIC Systems (for Algebraic Translation and Compilation for UNIVAC I and II).* Philadelphia Pa: Remington Rand UNIVAC.

Backus, J.W. 1954 January. The IBM 701 Speedcoding system. *JACM* 1 (1):4-6.

*Backus, John. 1954 May 21. Letter to J.H. Laning, Jr.

Backus, J.W. 1958 November. Automatic programming: properties and performance of FORTRAN systems I and II. In *Proc. Symp. on the Mechanisation of Thought Processes.* Teddington, Middlesex, England: The National Physical Laboratory.

Backus, John. 1976 June 10-15. Programming in America in the nineteen fifties—some personal impressions. In N. Metropolis, J. Howlett, Gian-Carlo Rota, Eds., *A History of Computing in the Twentieth Century* (Proc. of the International Conference on the History of Computing, Los Alamos, 1978), New York: Academic Press, in press.

Backus, John. 1978 August. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *CACM* 21 (8):613-641.

Backus, J.W. and Heising, W.P. 1964 August. FORTRAN. In *IEEE Transactions on Electronic Computers,* Vol. EC-13 (4):382-385.

Backus, John W. and Herrick, Harlan. 1954 May. IBM 701 Speedcoding and other automatic programming systems. In *Proc. Symp. on Automatic Programming for Digital Computers.* Washington DC: The Office of Naval Research.

Backus, J.W.; Beeber, R.J.; Best, S.; Goldberg, R.; Haibt, L. M.; Herrick, H.L.; Nelson, R.A.; Sayre, D.; Sheridan, P.B; Stern, H.; Ziller, I.; Hughes, R.A; and Nutt, R. 1957 February. The FORTRAN automatic coding system. In *Proc. Western Joint Computer Conf.* Los Angeles.

Baker, Charles L. 1956 October. The PACT I coding system for the IBM Type 701. *JACM* 3 (4): 272-278.

Belady, L.A. 1965 June 15. *Measurements on programs: one level store simulation.* Yorktown Heights NY: IBM Thomas J. Watson Research Center. Rep. RC 1420.

Böhm, Corrado. 1954. Calculatrices digitales: Du déchiffrage de formules logico-mathématiques par la machine même dans la conception du programme. In *Ann. di Mat. Pura ed Applicata* 37(4):175-217.

Bouricius, Willard G. 1953. December. Operating experience with the Los Alamos 701. In *Proc. Eastern Joint Computer Conf.* Washington DC.

Bright, Herbert S. 1971 November. FORTRAN comes to Westinghouse-Bettis, 1957. *Computers and Automation:* also *Ann. History of Computing* 1979 July.

Brown, J.H. and Carr, John W., III. 1954 May. Automatic programming and its development on MIDAC. In *Proc. Symp. on Automatic Programming for Digital Computers.* Washington DC: The Office of Naval Research.

Cocke, John and Schwartz, J.T. 1970 April. *Programming Languages and their Compilers.* (Preliminary Notes, Second Revised Version, April 1970) New York: New York University, The Courant Institute of Mathematical Sciences.

Cress, Paul; Dirksen, Paul; and Graham, J. Wesley. 1970. *FORTRAN IV with WATFOR and WATFIV.* Englewood Cliffs NJ: Prentice-Hall.

Dijkstra, Edsger W. 1976. *A Discipline of Programming.* Englewood Cliffs NJ: Prentice-Hall.

Grems, Mandalay and Porter, R.E. 1956. A truly automatic programming system. In *Proc. Western Joint Computer Conf.* 10-21.

Hoare, C.A.R. 1969 October. An axiomatic basis for computer programming. *CACM* 12 :576-580, 583.

*IBM. 1956 October 15. *Programmer's Reference Manual, The FORTRAN Automatic Coding System for the IBM 704 EDPM.* New York: IBM Corp. (Applied Science Division and Programming Research Dept., Working Committee: J.W. Backus, R.J. Beeber, S. Best, R. Goldberg, H.L. Herrick, R.A. Hughes [Univ. of Calif. Radiation Lab. Livermore, Calif.], L.B. Mitchell, R.A. Nelson, R. Nutt [United Aircraft Corp., East Hartford, Conn.] D. Sayre, P.B. Sheridan, H. Stern, I. Ziller).

*IBM. 1957. *Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704.* New York: IBM Corp. Form No. 32-0306.

Knuth, Donald E. and Pardo, Luis Trabb. 1977. Early development of programming languages. In *Encyclopedia of Computer Science and Technology,* Vol. 7:419-493. New York: Marcel Dekker.

*Laning, J.H. and Zierler, N. 1954 January. *A program for translation of mathematical equations for Whirlwind I.* Cambridge Mass.: MIT Instrumentation Lab. Engineering Memorandum E-364.

McCracken, Daniel D. 1961. *A Guide to FORTRAN Programming.* New York: Wiley.

Moser, Nora B. 1954 May. Compiler method of automatic programming. In *Proc. Symp. on Automatic Programming for Digital Computers.* Washington DC: The Office of Naval Research.

Muller, David E. 1954 May. Interpretive routines in the ILLIAC library. In *Proc. Symp. on Automatic Programming for Digital Computers.* Washington DC: The Office of Naval Research.

*Operator's Manual. 1957 April 8. *Preliminary Operator's Manual for the FORTRAN Automatic Coding System for the IBM 704 EDPM.* New York: IBM Corp. Programming Research Dept.

Organick, Elliot I. 1963. *A FORTRAN Primer.* Reading Mass.: Addison-Wesley.

*Perlis, A.J.; Smith, J.W.; and Van Zoeren, H.R. 1957 March. *Internal Translator (IT): a compiler for the 650.* Pittsburgh: Carnegie Institute of Tech.

*Preliminary Report. 1954 November 10. *Specifications for the IBM mathematical FORmula TRANslating system, FORTRAN.* NY: IBM Corp. (Rep. by Prog. Res. Group, Appl. Sci. Div., IBM. Distrib. to prospective 704 customers and other interested parties. 29 pp.)

*Proposed Specifications. 1957 September 25. *Proposed Specifications for FORTRAN II for the 704.* (Unpublished memorandum, Programming Res. Dept. IBM.)

*Remington Rand, Inc. 1953 November 15. *The A-2 compiler system operations manual.* Prepared by Richard K. Ridgway and Margaret H. Harper under the direction of Grace M. Hopper.

Rutishauser, Heinz. 1952. Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen. In *Mitteilungen aus dem Inst. für angew. Math. an der E. T. H. Zürich.* Nr. 3. Basel: Birkhäuser.

Sammet, Jean E. 1969. *Programming Languages: History and Fundamentals.* Englewood Cliffs NJ: Prentice-Hall.

Sheridan, Peter B. 1959 February. The arithmetic translator-compiler of the IBM FORTRAN automatic coding system. *CACM* 2 (2):9-21.

*Schlesinger, S.I. 1953 July. *Dual coding system.* Los Alamos NM: Los Alamos Sci. Lab. Rep. LA1573.

Zuse, K. 1959. Uber den Plankalkül. In *Elektron. Rechenanl.* 1:68-71.

Zuse, K. 1972. Der Plankalkül. In *Berichte der Gesellschaft für Mathematik und Datenverarbeitung* 63, part 3. Bonn. (Manuscript prepared in 1945.)