Cress, Paul, Dirksen, Paul, and Graham, J. Wesley (1970). *FORTRAN IV with WATFOR and WATFIV*. Englewood Cliffs, New Jersey: Prentice-Hall.

Dijkstra, Edsger W. (1976). *A Discipline of Programming*. Englewood Cliffs, New Jersey: Prentice-Hall.

Grems, Mandalay, and Porter, R. E. (1956). A truly automatic programming system. In *Proc. Western Joint Computer Conf., San Francisco*, pp. 10–21.

Hoare, C. A. R. (1969) October. An axiomatic basis for computer programming. *CACM* **12**(10): 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, and 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 Trabb Pardo, Luis (1977). Early development of programming languages. In *Encyclopedia of Computer Science and Technology*. Vol. 7, pp. 419–493. New York: Dekker.

*Laning, J. H., and Zierler, N. (1954) January. *A Program for Translation of Mathematical Equations for Whirlwind I*. Cambridge, Massachusetts: 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, D.C.: 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, D.C.: 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, Massachusetts: Addison-Wesley.

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

*Preliminary Report (1954) November 10. *Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN*. New York: IBM Corp. (Report by Programming Research Group, Applied Science Division, IBM. Distributed 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 Research 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*. No. 3. Basel: Birkhäuser.

Sammet, Jean E. (1969). *Programming Languages: History and Fundamentals*. Englewood Cliffs, New Jersey: Prentice-Hall.

*Schlesinger, S. I. (1953) July. *Dual Coding System*. Los Alamos, New Mexico: Los Alamos Scientific Lab. Los Alamos Report LA 1573.

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

Zuse, K. (1959). Über den Plankalkül. *Elektron. Rechenanl.* **1**: 68–71.

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

TRANSCRIPT OF PRESENTATION

JAN LEE: Our second session this morning, and the first session dealing explicitly with a language, is to be the paper by John Backus on the History of FORTRAN I, II, and III.

I had an interesting occurrence just before I left my office this week. I had a letter from Taiwan, believe it or not, saying, "In 1959 you wrote a program, and I have a customer who's very interested in this topic. Could you send me the source listing, etc., etc., of this?'' And believe it or not, I still do have the source listing. It turned out to be a language which, as Grace would have said earlier on, died, and I was looking back and saying, ''Gee, 1959—why wasn't I programming in FORTRAN at that time? What was I doing?'' And I realized of course that I was one of those people who were steadfastly resisting this movement into the algebraic field, and I guess I really didn't understand that you could do things with letters and alphabets and so on and so forth. So, amongst many many others, I was behind the times. And it keeps coming back to haunt me, which worries me.

This first speaker this morning is the man who was trying to educate the rest of us and say, ''This is the way to go.'' John mentioned to me, when we were discussing how I should introduce him, that there was only one real thing to say—and I'll say that at the end, as we did with Grace. But I wanted to try and introduce him in two ways: one was —how would we have introduced him in 1955, if this were 23 years ago—what would we have said? At that point I guess we would say: Well, here's an IBM-er; he's a programmer, a very lazy programmer who feels that he wants to make everything much easier for everybody else; holds a Master's Degree in Mathematics from Columbia, and has currently moved from the Pure Science Division of IBM to the Applied Science Division of IBM, which as an engineer I feel means he saw the light about that time.

Since that time things have progressed 23 years, and so I say, ''Okay, now where are we? How would I introduce him in 1978?'' Well, he's an IBM Fellow, interested in programming languages, and is still a lazy programmer—John Backus.

JOHN BACKUS: Well, Grace Hopper is a hard act to follow, but that's my fate. Since I'm sure everyone has studied my paper very carefully already, I can skip parts of it, and that's fortunate because, of course, it would be impossible to cover everything in it.

I'd like to begin by getting some idea of the make-up of this audience. And to do that I'd like everyone who was involved in programming before 1960 to raise their hand. 1960 —Wow! Okay, let's try 1956. My gosh! 1954. Getting smaller. Okay—I'll stop there. Because that's the point where I'm going to begin—where Grace Hopper left off. I'd like to try to add something to what Grace Hopper has already said about what things were like in 1954, and the attitudes that prevailed then.

It's really difficult for a programmer today, who wasn't involved then, to realize how ignorant we were then, and how primitive the ideas and tools were that we felt were really advanced and sophisticated stuff. In late 1953 computers were pretty crazy things. They had very primitive instructions and extremely bizarre input-output facilities. Some were better than others. I think Grace had the benefit of rather good input-output facilities, but the rest of us didn't.

So, overcoming the difficulties of these computers and cramming programs and data into little tiny stores made programming really a black art. So each programmer had his own inscrutable tricks for making programs as fast and as complicated as possible. Some so-called automatic programming systems in 1953, would you believe, did the wonderful thing of providing decimal input or decimal input and output, and some did other wonderful things like providing symbolic addresses.

The most popular automatic programming systems, as Grace Hopper said, were interpretive systems. These just converted the peculiar host machine into a different synthetic machine which had floating point and improved I/O facilities and sometimes indexing. Could I have the first slide [Frame 1], please?

Name	Computer	Place developed
DUAL	IBM 701	Los Alamos
SHACO	IBM 701	Los Alamos
Summer Session	WHIRLWIND	M.I.T.
† David Wheeler's	ILLIAC	Univ. of Illinois
†SPEEDCODING	IBM 701	IBM New York

Frame 1. 1953 interpretive systems. These systems were all operating in 1953. Each provided a synthetic computer with floating point, indexing† and improved I/O.

Here are some 1953 interpretive systems that were running that I know of. Probably nothing is complete but I've done the best I can. So each one made its machine look like a different machine. It had an order code—you know, a machinelike order code, but it now could do floating point arithmetic, and things like that. So it made programming easier, but you were still really programming for a machine. So, in addition to these interpretive systems, as far as I know there were three compilers operating in 1953—there they are [in Frame 2]. I gather from what was said earlier, that the date for A-2 was earlier than that on the slide, but these dates are taken from an article by Knuth and Trabb and that was the date they gave, so I'm not responsible for that. We'll take a look at some sample programs for these systems in a minute. But let me just say that first of all, note that the AUTOCODE system was a fixed point system. It didn't provide floating point, and it was really quite machine-dependent, as we'll point out a little more when we look at a program for it.

The A-2 language was not algebraic, but it did provide floating point operations. Its input language was not what we would call sophisticated nowadays. And in fact, its input language was somewhat more complicated and difficult to use than the machinelike languages of the interpretive systems.

Sometime in late 1954 or early 1955 the A-2 system acquired a pseudo-code that resembled the input languages of the interpretive systems that were listed on the previous slide.

The Laning and Zierler system was—I believe now everyone recognizes it to be—the world's first algebraic compiler. It was operating in May 1953, and possibly earlier. Some communications I've had from what used to be called the "Instrumentation Laboratory" indicated that it was perhaps earlier than that. Its input language was simple but elegant, and it compiled algebraic expressions into programs that were then interpreted by a floating point interpretive system.

(a) Alec Glennie's AUTOCODE, Manchester Mark I.
Not algebraic, very machine dependent, fixed point.
Early use: Sept. 1952

(b) Grace Hopper's A-2, UNIVAC.
Not algebraic, machine dependent, floating point.
Early use: late 1953 (A-0 & A-1 earlier)

(c) Laning & Zierler's system, WHIRLWIND.
Algebraic, machine independent, interpreter executes compiled floating point instructions.
Early use: May 1953.

Frame 2. Compilers operating in 1953 (and earlier).

```
c@VA t@IC x@½C y@RC z@NC
INTEGERS +5 → c
   →t
  +t     TESTA Z
  −t
         ENTRY Z
SUBROUTINE 6 → z
  +tt → y → x
  +tx → y → x
+z+cx CLOSE WRITE 1
a@/½  b@MA  c@GA  d@OA  c@PA  f@HA  .@VE  x@ME
INTEGERS +20 → b  +10 → c  +400 → d  +999 → e  +1 → f
LOOP 10n
  n → x
+b − x → x
  x → q
SUBROUTINE 5 → aq
REPEAT n
  +c → i
LOOP 10n
  +an SUBROUTINE 1 → y
  +d − y TESTA Z
  +i SUBROUTINE 3
  +e SUBROUTINE 4
      CONTROL X
      ENTRY Z
  +i SUBROUTINE 3
  +y SUBROUTINE 4
      ENTRY X
  +i − f → i
REPEAT n
ENTRY A CONTROL A WRITE 2 START 2
```

Frame 3. An AUTOCODE program.

Now we can see some of the programs for these things. [Frame 3 shows an AUTOCODE program.] I'm sure you can't read that very well. It looks somewhat symbolic, but it really is more machine-dependent than it looks. Like "Test A" really refers to a particular machine register. "Close Right 1" means that that is the end of a subroutine, called "Subroutine 1," which is referred to later. I can't really explain these programs at all. I just wanted you to get a feel for what they looked like. This program is for a nonsense algorithm that Knuth and Trabb programmed for lots of early languages, to compare them, called the TPK Algorithm.

And here [in Frame 4] you can see the A-2 program that they coded for that same TPK Algorithm. The horizontal lines separate unit operations, and some of them you can see —the ones beginning "GM" are "generators" that Dr. Hopper mentioned; the single line operations are floating point instructions; and you see a couple of entries starting with OWNCO, and that is Machine Code that was necessary to use; the point I am trying to illustrate is that A-2 does not have a uniform input language. There's quite a variation in the formats of the unit operations; some take one line, some take 2, some take three, and some take four.

Now we can move on to [Frame 5] the TPK Algorithm for Laning and Zierler's system,

```
GMI000   000002    AA0040   038038    YT0036   038000    4RG000   000007
ITEM01   WS.000    AS0004   038040    GMM000   000001    5RG000   000006
SERV02   BLOCKA    0WNΔC0   DEΔ003    000194   200220    6RG000   000007
1RG000   000000    K00000   K00000    1RG000   011000    1CN000   000002
GMM000   000001    F00912   E001RG    GMM000   000001    2CN000   000014
000180   020216    000000   Q001CN    000222   200196    1RS000   000036
1RG000   001000    1RG000   008040    1RG000   012000    2RS000   000037
AM0034   034040    1CN000   000010    ALL012   F000Ti    0WNΔC0   DEΔ002
RNA040   010040    GMM000   000001    1RG000   013036    810000   820000
APN034   012038    000188   020238    2RG000   000037    900000   900000
AM0002   038038    1RG000   009000    3RG000   000006    1RG000   014000
                                                          R̸ØENDΔ   INFO.R̸
```

Frame 4. A-2 "TPK" ALGORITHM. [From Donald E. Knuth and Luis Trabb Pardo, The early development of programming languages. In *Encyclopedia of Computer Science and Technology*, Vol. 7, pp. 419–493. New York: Dekker, 1977.]

and we see that their language is algebraic. It has single subscripts which are denoted by the vertical bar, so that $v|j$ in the fourth line is V_j. Subroutines are indicated by the capital "F" with superscripts, and the superscripts are the "catalog number" of the subroutine, so you couldn't name subroutines except in that way. And it lacked a looping statement, a FOR statement, or what-have-you. But this language certainly is elegant by comparison with the earlier ones, and when you realize that it was running very early in 1953, if not earlier, you can get some idea of the accomplishment that it represented. And yet of course, as Grace Hopper said, the Establishment completely ignored this system for some peculiar reason.

In addition to the compilers that were actually operating in 1953, Heinz Rutishauser in Switzerland and Corrado Böhm in Italy had both published descriptions in 1952 of simple algebraic languages and their compilers. They really published the details of their compilers. And in fact, Böhm's compiler was actually written in its own language; the first example of such a thing.

And even earlier, Konrad Zuse in Germany had described in 1945 what was perhaps the most sophisticated programming language that was to emerge up to the mid 1960s. It was really a dilly. A very powerful language with data types and all kinds of things. He wrote many chess programs in that language at that early time.

For a discussion of these systems that I've shown, and other early systems—there are

```
    v|N = ⟨input⟩,                 CP 3,
    i = 0,                         z = 999,
1   j = i+1,                       PRINT i,z.
    a|i = v|j,                     SP 4,
    i = j,                     3   PRINT i,y.
    e = i-10.5,                4   i = i-1
    CP 1,                          e = -0.5-i,
    i = 10,                        CP 2,
2   y = F^1(F^11(a|i))+5(a|i)^3,   STOP
    e = y-400,
```

Frame 5. Laning and Zierler "TPK" algorithm. [From Donald E. Knuth and Luis Trabb Pardo, The early development of programming languages. In *Encyclopedia of Computer Science and Technology*, Vol. 7, pp. 419–493. New York: Dekker, 1977.]

others—I recommend the article I mentioned by Knuth and Trabb in the *Encyclopedia of Computer Science and Technology*.

So I hope I've added a little something to what Grace Hopper said about the primitive state of affairs at the beginning of 1954. That was the point at which the story of FORTRAN begins.

About December 1953 I wrote a letter to my boss, Cuthbert Hurd, proposing that IBM should develop a practical automatic programming system for what was then the recently announced 704. He agreed, even without an argument. I don't know why Grace Hopper had so much trouble with her management! [Laughter] But we didn't have any at all. They kept backing us up for years and years without anything to show for it. But anyway, our plan was to produce a system that would output efficient object code. We felt that this was essential if the system was to be widely used for the big problems of the aerospace industry and that, of course, was the principal customer for the 704 at that time.

Most of the early systems we've seen, and many others, had hidden a lot of gross inefficiencies by virtue of the fact that most computing time was being spent in floating point subroutines. Because of that you could get away with a lot of clumsy treatment of looping and indexing, and references to arrays that escaped unnoticed because of all the time that was chewed up by floating point subroutines. But now we were confronted with the 704 which, for the first time, had built-in floating point and indexing, and we knew that this would make our goal of producing efficient object code very difficult because there was just nowhere to hide inefficiencies. You couldn't do clumsy calculations of subscript combinations and get away with it in that machine.

So by 1954, the FORTRAN group had become actually a group. It consisted of Irving Ziller—this is how he looks 24 years after the fact [Frame 6]; Harlan Herrick [Frame 7], and Bob Nelson [Frame 8]. Bob was actually hired to do technical typing for our group,

Frame 6. Irving Ziller.

Frame 7. Harlan Herrick.

but he and Irv Ziller were soon involved in planning and programming one of the most complex sections of the compiler. Nelson was later made an IBM Fellow, in fact, for his work on Virtual Memory, that we've heard mention of before! It's a rather broad term! And I was the fourth member of the group—so you can see me as I am. And I was its so-called "Manager." [See photo on p. 74.]

The early thinking of our group about the system that was to become FORTRAN is indicated by a paper that Herrick and I gave at the ONR symposium in May of 1954 which

Frame 8. Bob Nelson.

"[The programmer] would like to write $\Sigma a_{ij} \cdot b_{jk}$ instead of the . . . instructions [for] this expression."
". . . a programmer might not be considered too unreasonable if he were willing only to produce the formulas for the numerical solution of his problem and perhaps a plan showing how the data was to be moved from one storage hierarchy to another and then demand that the machine produce the results . . ."

Frame 9. From a May, 1954 article by Backus and Herrick. [From John W. Backus and Harlan Herrick, IBM 701 speedcoding and other automatic coding systems. In *Proc. Symp. on Automatic Programming for Digital Computers*. Washington D.C.: ONR, May 13–14, 1954.]

was organized by Grace Hopper. Her name seems to keep popping up in all these things. Here are some excerpts from that paper [Frame 9]. This shows that our plans for algebraic input were, in May of 1954, a bit more sophisticated than the Laning and Zierler language in that we had double subscripts and things like that. In fact I'd say that the goal suggested in the second paragraph about handling storage has yet to be satisfactorily achieved.

That brings up the questions of the influence of Laning's work on the FORTRAN language. A few years after FORTRAN was finished I was asked where we got the idea for algebraic input. Since I have a very poor memory, I recalled erroneously that we had begun work later than we actually had and that we had gotten the idea of algebraic input from seeing Laning and Zierler's system, and I went on believing this story for many years. But recently Dr. Laning was good enough to send me a copy of the letter I sent him in 1954 asking to see his system, and this letter—plus some recent discussions with Irv Ziller—made clear what really happened. The letter, in fact, makes clear that we first heard of Laning's work at the same ONR symposium at which those remarks appeared that I just showed you, and so we had already been thinking about algebraic input, and a fair number of other things. And so we can only conclude that we somehow had the idea for algebraic input independently of them, even though Laning and Zierler already had done their pioneering work and had their compiler running. As was common in those days news didn't get around very well.

So, before taking up the chronological account of the development of FORTRAN, I'd like to emphasize one other prevailing attitude in the period of 1954–1957, when we were working; and this was the firm belief that almost all programmers held—I guess JAN Lee was one of them—that no automatic programming system could produce code that was even comparable in efficiency to hand-coded programs. This belief came from their experience in coding for small, badly designed computers—that was truly a difficult task—and their experience with clumsy and slow automatic programming systems, of which there are quite a few. I was the author of one of them.

By the spring and summer of 1954, the design of the FORTRAN language was well underway. Most of the language was designed by Herrick, Ziller, and myself, but most of the input–output language was designed, mostly later on, by Roy Nutt, and we'll see a picture of him later on. Roy is one of the grand programmers of the profession. There's a myth—I don't know whether it's really true about him or not—that he sits down at the keypunch and programs. You'll have to ask him though whether that's true or not, . . . but he wrote an awful lot of programs for us.

There's very little I can say about how the language was designed. It never seemed like much of a problem to us, for some reason. Very early in our fooling around we had the

"Since FORTRAN should virtually eliminate coding and debugging[!] . . ."
". . . an automatically coded problem . . . will be executed in about the same time that would be required had [it] been laboriously hand coded."
"Furthermore, after an hour course in FORTRAN notation, the average programmer can fully understand . . . a procedure stated in FORTRAN language . . ."
". . . since each such [IBM] calculator should have a system similar to FORTRAN . . ."

Frame 10. Excerpts from FORTRAN *Preliminary Report*, Nov. 1954.

notion of using algebraic expressions. It didn't seem to be very original. Subscripted variables didn't seem to be very original. And assignment statements. Shortly after that, Herrick, I believe it was, proposed the DO statement or "formula" as we used to call statements in those days. And the other features of the language just arose from the need to make some machine function available. Many statements were machine-dependent—things like IF DIVIDE CHECK or READ DRUM were popular items. And other statements arose from just writing test programs and finding we didn't have enough statements to write the programs, so we'd make up one and add it to the language.

So we always regarded the real problem to be that of consistently producing object programs about as efficient as hand coded ones, because we figured that if we didn't do that, no one would use the system.

By November of 1954 our group had produced a paper that really completely described the FORTRAN language and, we'll see a few excerpts from that report [Frame 10]. As you can see from the first quote, we were not exactly a group of pessimists, and the third quote may have been more of a comment about the quality of programmers then than about the ease of learning FORTRAN. And here [Frame 11] is part of the description of expressions in that paper, just so you'll get some idea of style. I omitted the two longer ones, of course. And here [Frame 12] are some of the many proposed additions to the system that are suggested at the end of the paper. There were quite a few of them. The description of the proposed function definition facility is badly written, but it really does describe basically what later appeared in FORTRAN II in 1958.

The last [item on Frame 12] is "summation." We had planned to put in a summation operator that would take the index of summation, and the bounds, and the expression to be summed.

So the language described in that 1954 preliminary report was very close to the actual language that came out as FORTRAN. It had several features that were later dropped or changed. For example, the DO formula, as it was called, could specify a distant sequence of formulas to be iterated on. It didn't have to follow the DO statement. It could also spec-

(i) Any constant or variable is an expression.
(ii) If E is an expression not of the form +F or −F, then +E and −E are expressions.
(iv) If E is an expression, so is (E).
(vi) If E and F are expressions, so is E × × F. [sic]

Frame 11. Four lines (of six) from the "Formal description" of "expressions" from the *Preliminary Report*, Nov. 1954.

Begin Complex Arithmetic
Sort the vectors on tape N . . .
General Function: . . . the programmer [could] specify the formula numbers of the formulas describing his function and the arguments to be used in a given instance. The value [would be obtained] having substituted the specified arguments for the original arguments . . .
Summation

Frame 12. Excerpts from "Future additions to the FORTRAN system" from the *Preliminary Report*, Nov. 1954.

ify the statement that was to follow the last iteration separately. Expressions could be mixed mode, floating and fixed. And that got changed, I think, just because we didn't like the rules as to what happened with mixed mode expressions, so we decided, "Let's throw it out. It's easier." And it did leave in the principal uses of mixed modes—like even in FORTRAN I you could have an integer exponent and integer function arguments in a floating point expression. In IF formulas in this—Knuth calls it FORTRAN 0—you could use an inequality sign between two expressions and then give two formula numbers; one for the true case of the inequality, and one for the false case.

Then there was a RELABEL formula that's very hard to describe; it caused the subscripts of a particular array that you mention to be reinterpreted in a rotated fashion, and the purpose of that was to simplify calculations on arrays that were too large to fit in the store, so that the same stated calculation would work and you could read in the new row, wherever the current new row was supposed to go, and then rotate one place, and the whole calculation would still work. But it was too hard to describe, and much too hard to implement, so that got dropped.

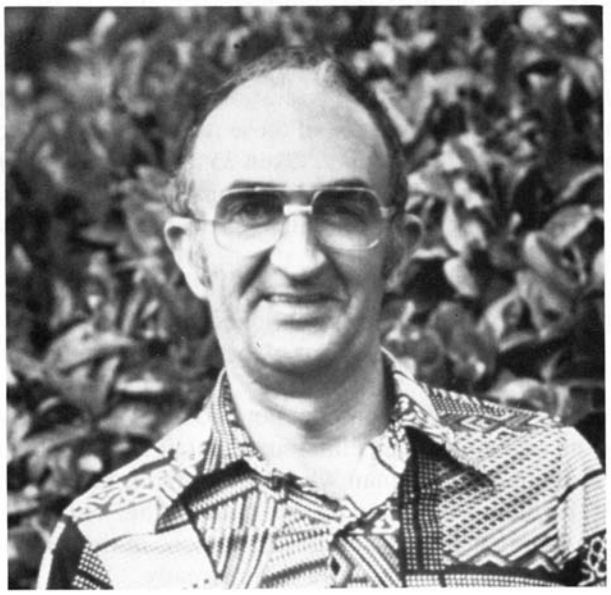
FORMAT statements were lacking then, but they appeared as the first item in the section on future additions to the FORTRAN system, part of which I showed you. And apparently already at this time, we had been talking to Roy and that was probably his suggestion.

It's perhaps interesting to note that executable statements in FORTRAN 0 were called "formulas" but declarations such as EQUIVALENCE and DIMENSION statements were called "specification sentences."

The final section of this paper discusses FORTRAN programming techniques needed to produce optimum object programs. It discusses how the system identifies identical subexpressions and points out that DO loops would be treated optimally, whereas loops formed by IF's and GOTO's would not be.

In late 1954 and early 1955 Herrick, Ziller, and I gave five or six talks in various places about our plans for FORTRAN to groups of IBM customers who had ordered 704s. We described the language and our plans for producing optimized object programs. We estimated then that the compiler would be done in about six months, and this six-month interval was to remain the estimated time to completion for the next two years. And of course most of our listeners were extremely skeptical about all of our claims. Other groups had described in rosy terms their own automatic programming systems, and these had turned out to have a lot of restrictions and oddities in real life that had not been mentioned in the early descriptions. And in particular, many of those systems had been dreadfully slow.

So our claims were even more outlandish than those of others, and thus the skepticism



Frame 13. Roy Nutt.

they aroused was not really unreasonable. Our claims that we would produce efficient object code were especially disbelieved.

But one of these talks to customers had a very fortunate and crucial result. After our talk at United Aircraft, Walter Ramshaw, who was the head of that shop, agreed that Roy Nutt [Frame 13] would become a regular member of our group and contribute his expertise on input-output and assembly programs. As many of you know, Roy wrote the SHARE assembly program called SAP, and a lot of other utility programs for SHARE. All the time he worked with us he remained an employee of United Aircraft, and he used to travel from Hartford to New York two or three times a week from early 1955 up until early 1957.

There is no way to know what influence the FORTRAN preliminary report had on other groups. But some other groups produced algebraic compilers which were running before we distributed FORTRAN in April of 1957. They were able to do this in part because they had more modest goals. Mainly, they were not attempting to optimize their object programs. But we did have the impression that the fairly wide distribution of our preliminary report, plus the general awareness of many programming groups that we were involved in a fairly big effort and were pretty serious about it—that this provided a stimulus to some other groups that they might not have had otherwise.

In any event, the IT compiler at Purdue—it was later carried on at Carnegie Tech—the MATH-MATIC Compiler at Sperry Rand, and the BACAIC Compiler at Boeing were done at various points after late 1954. It's hard to be precise about when each of these began running, just as it's hard to be precise about when FORTRAN began running. For example, FORTRAN ran in various stages, and then the complete system ran for some unknown time before we distributed it. The only really definite date is when we distributed it.

In any case, whatever the order in which these systems began running, it's certainly clear that those three systems were running before we distributed FORTRAN.

The most important of these compilers was the IT compiler for the IBM 650 that was done by Alan Perlis and his colleagues. It was begun in the fall of 1955 and since, like the others, it was less ambitious than FORTRAN, it was running about a year later. After FORTRAN and IT were both in operation for a while, Alan Perlis used to give some talks that really annoyed me. The idea of these talks was that he couldn't understand why those clods working on FORTRAN had taken 25 man-years to produce a compiler, since one of his graduate students had written an IT compiler in a single summer. He apparently didn't understand the problems of optimization in those days. I was annoyed only because I thought he did!

In any case, whatever the influence the FORTRAN preliminary report and our project had on the other projects that I mentioned, if it had any, it is clear that they didn't influence our design because we had our plans quite firm in November of 1954, before they had begun.

Work on the FORTRAN compiler got underway in January or February of 1955, and my paper for the conference goes into some detail about its construction, over, I might say, the objections of our chairman who doesn't think that's a proper subject for this conference. And it references other papers with further details. But now I would just like to sketch very briefly what each of the six sections of the compiler did and show you its principal architects if we haven't seen them already.

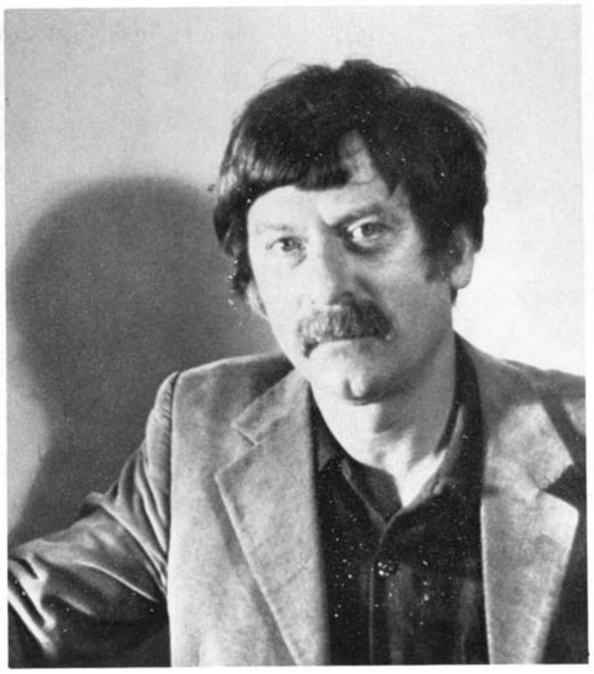
The main point to understand about the way the thing was put together was that each section was a separate and autonomous project in itself of one or two people. Each section really was an individual enterprise; each had its proprietor or proprietors and was designed, planned, and programmed by a very small group. That planning often involved the invention of a number of new techniques by the proprietors, and many of those techniques, of course, played an important role in future compilers. The proprietors of each section would consult with the neighboring sections and by doing so on and off for a long time, they gradually converged on specifications for their input from the preceding section, and the output they would give to the next section.

Section 1 read the entire source program. In that sense, the compiler was one-pass. It compiled instructions for the arithmetic expressions, and filed all the other information in tables for use by later sections.

Peter Sheridan [Frame 14] did the compilation and optimization of expressions in Section 1. His output rearranged the computation so that the recomputation of common subexpressions and unnecessary fetches and stores were avoided. That work as we'll discuss later often changed the way expressions were evaluated in a radical and surprising fashion. Roy Nutt handled the I/O statements in Section 1—the main thing that he had to do was replace the I/O "lists of quantities" that were involved by nests of DO formulas, which were then treated by the rest of the compiler in a regular fashion. Herrick did the table formation work in Section 1.

Section 2 optimized the treatment of DO formulas and references to arrays. To do so, it had to analyze the structure of the entire source program. There is not enough time to discuss the powerful and complex analysis of cases that it used to produce code. It yielded virtually optimal use of indexing and looping. Section 2 is the largest and one of the two most complex sections of the compiler. It was planned and programmed by Bob Nelson and Irving Ziller.

The degree of optimization they achieved was really not equalled again in subsequent compilers until the mid-60s when the work of Fran Allen and John Cocke began to be used


Frame 14. Peter Sheridan.

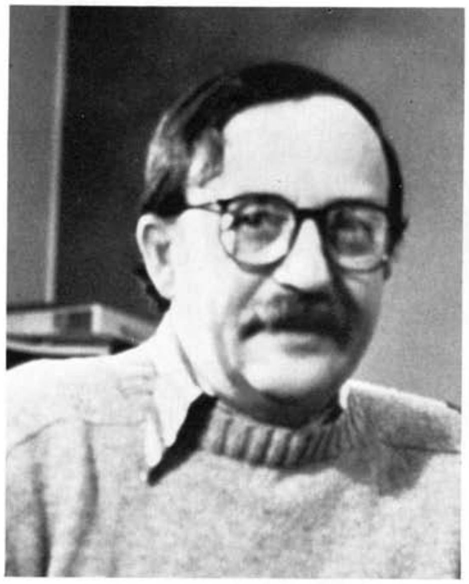
in fairly sophisticated optimizing compilers. They really did an incredible job, even taking out single instructions that you wouldn't believe could have been taken out of a program.

Originally Ziller and Nelson had planned to produce code which used only the three index registers for the 704, but as things progressed it began to appear that that was going to be extremely difficult to do in an optimal way. So at that point, I proposed that they should produce code for a mythical 704 that had an unlimited number of index registers, and that we would somehow produce later sections that would convert that program into one for a three-index-register machine.

This suggestion resulted in two new sections of the compiler, but before they could begin we had to have a Section 3 which consolidated all the information from Sections 1 and 2, and was a fairly involved program. This was programmed by Dick Goldberg [Frame 15].

Section 4 was programmed by Lois Haibt [Frame 16]. It made a statistical analysis of the frequency of execution of the different parts of the source program by simulating the execution of the source program. It also collected information about the usage of the fictitious and unlimited index registers.

The proprietor of Section 5 was Sheldon Best [Frame 17] who was loaned to us by Charles Adams at MIT. (Sheldon was really the same age as the rest of us, but that is the only picture we have that was taken when we were actually working on the project.) Section 5 did the actual transformation of a program with an unlimited number of index registers into one using only three. The index register allocation procedure that Sheldon devised for Section 5 was at least as optimal as any procedure developed during the next 15 years, and it probably is in fact optimal. Nobody has shown that it isn't. On the other hand, nobody has proved that it is—although his straight-line allocation procedure has


Frame 15. Dick Goldberg.

Frame 16. Lois Haibt.

Frame 17. Sheldon Best.

been proved to be optimal, and at the time, he had a fairly convincing argument that it was. His work was the basis of a lot of later work on the register allocation problem.

Finally, Roy Nutt wrote Section 6. This assembled the final program into relocatable binary. It also produced the storage map of the program and data, and obtained the library routines that were needed, including routines that interpreted the FORMAT statements in the program. Roy also wrote the FORMAT interpreter and all of the I/O program. The assembler took advantage of the special nature of its input and was about ten times faster than the standard SHARE assembler—SAP.

By the summer of 1956, large parts of the compiler were working; an algebraic subsystem was in productive use at United Aircraft from the summer of 1956 until the full system was distributed. From the late spring of 1956 until early 1957 the pace of debugging of the system was really intense. We would rent rooms at the Langdon Hotel across the street from where the computer was in New York, and we'd sleep there a little bit during the day and then stay up all night to get as much time on the machine as possible. We usually had it all to ourselves then. It was a really exciting period because by late summer and early fall we were beginning to get fragments of compiled programs out of the system, and we were often astonished at the code it produced. It often transformed the source code so radically that we would think it had made an error, and we'd study the program carefully, and finally realize it was correct. Many of the changes in the computation were surprising, even to the authors of the section responsible for them. But the changes really were effective in giving efficient programs, and they would, for example, make a major rearrangement of the program just to save a single instruction. For example, control transfers would appear in the program that corresponded to no control statement in the original program. And a given DO statement would sometimes produce no instructions if you put it in one context, and if you took the same identical DO statement and put it in a different context, it would produce a whole lot of instructions in a lot of different places in the object program.

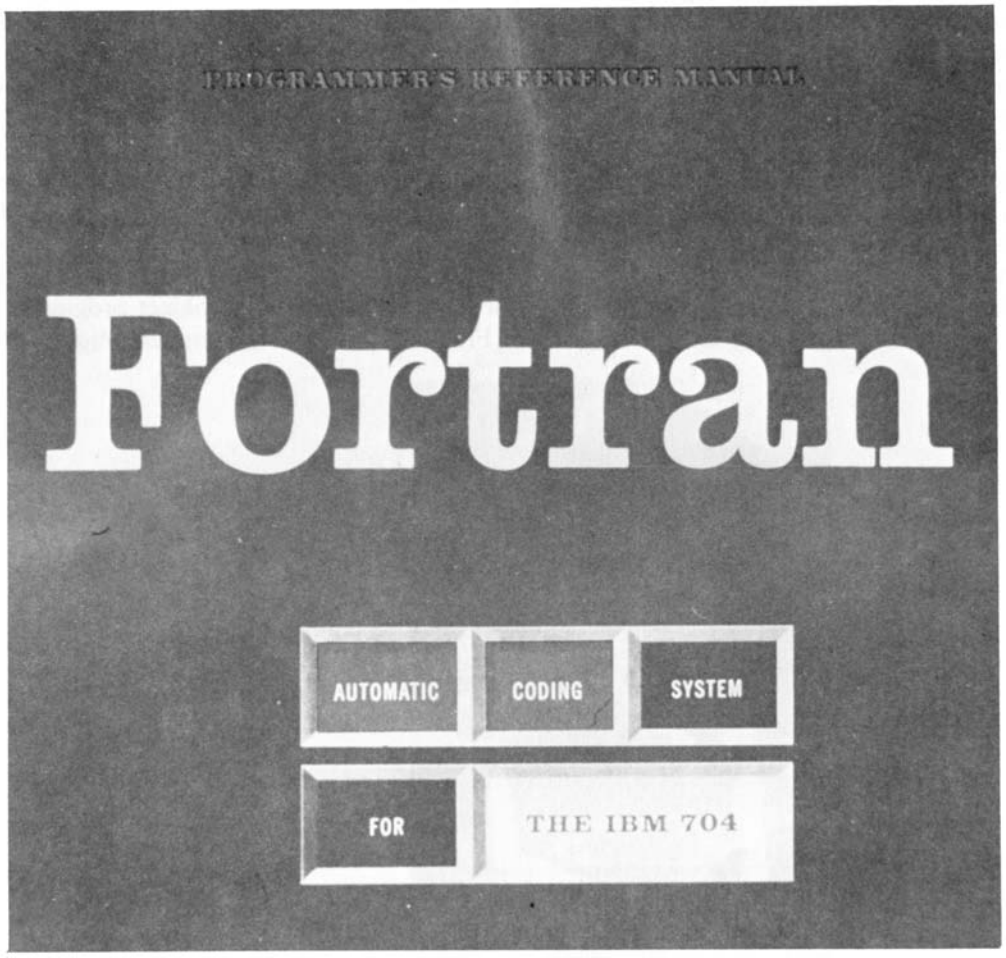
So—in the summer of 1956 David Sayre [Frame 18], a crystallographer, rejoined our

Frame 18. David Sayre.

group. He had briefly consulted with Sheldon Best about Section 5 earlier. He began writing the FORTRAN programmer's reference manual. This manual stood for some time as a unique example of a clear, concise, and accurate description of a programming language, and it may still stand as such. It appeared in a glossy cover and handsomely printed in October of 1956, and there's a shot of the cover [Frame 19]. Here is the Table of Contents [Frame 20], and here's a sample program that appeared on Page 1 of Chapter 1 [Frame 21], to show you the flavor of things. Here is a description of arithmetic formulas [Frame 22]. You can see that it had very wide margins with headings out in the margin. Here is the description of DO formulas [Frame 23: see Fig. 3 in Paper]. It followed this general pattern of having a description of the general form of a statement and then some examples, and then text giving the details. And finally, there is a longer program that appeared toward the end of the manual [Frame 24], an example, showing various features.

The entire Reference Manual was only 51 pages long. That's some kind of record, I think, and as you noticed, the pages even had wide margins.

I'm afraid I haven't really done justice in this brief talk to the work my friends did on the



Frame 19. Cover of first FORTRAN Manual. [Courtesy of International Business Machines Corporation.]

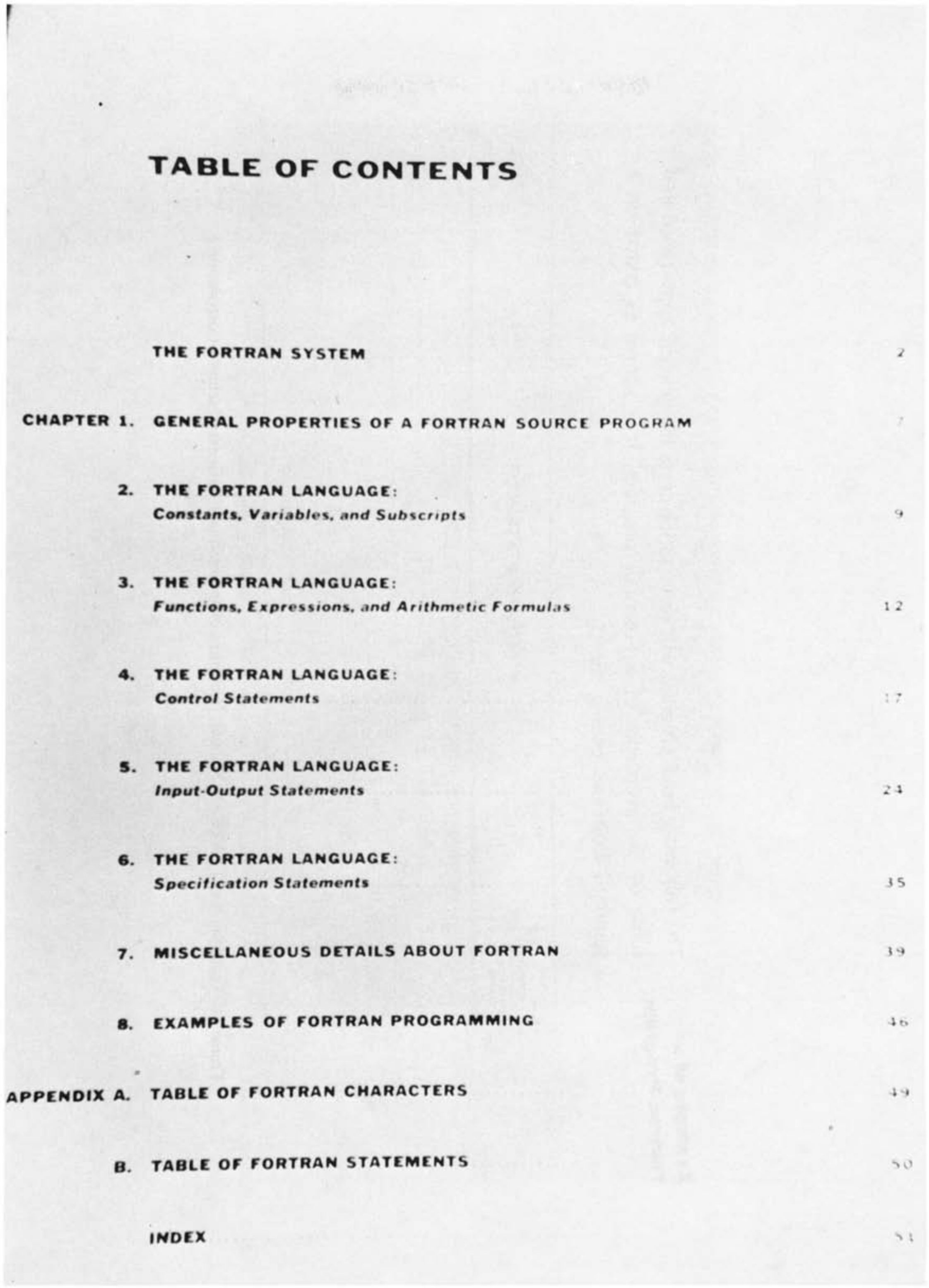
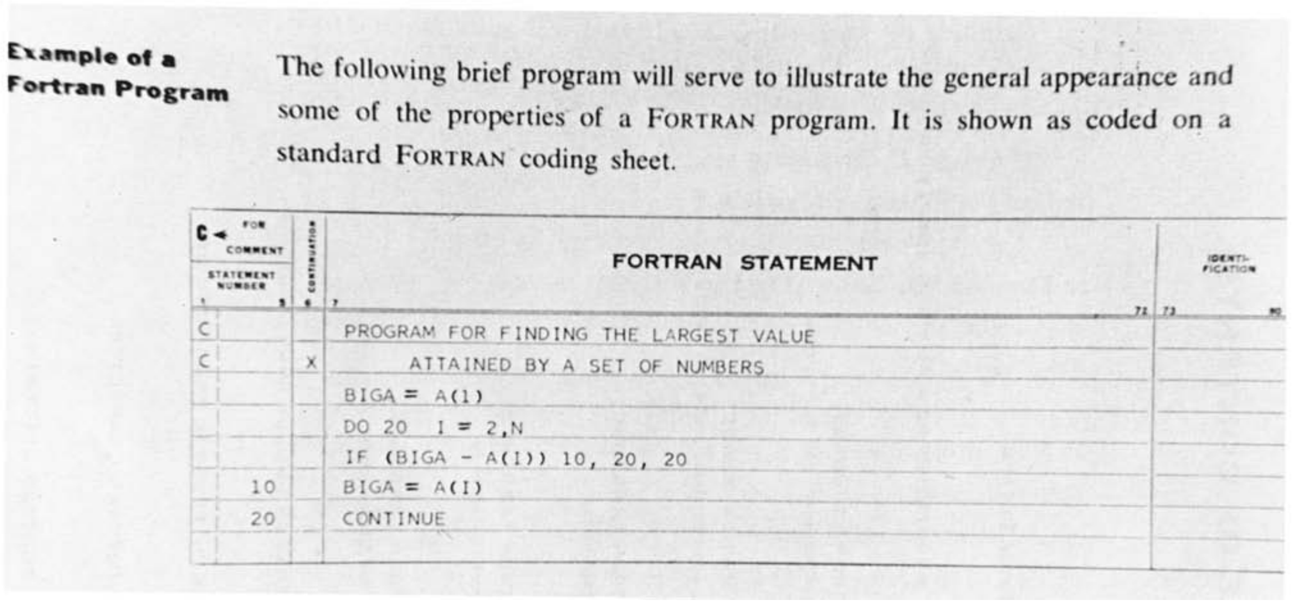


TABLE OF CONTENTS

Frame 20. From the first FORTRAN Manual. [Courtesy of International Business Machines Corporation.]

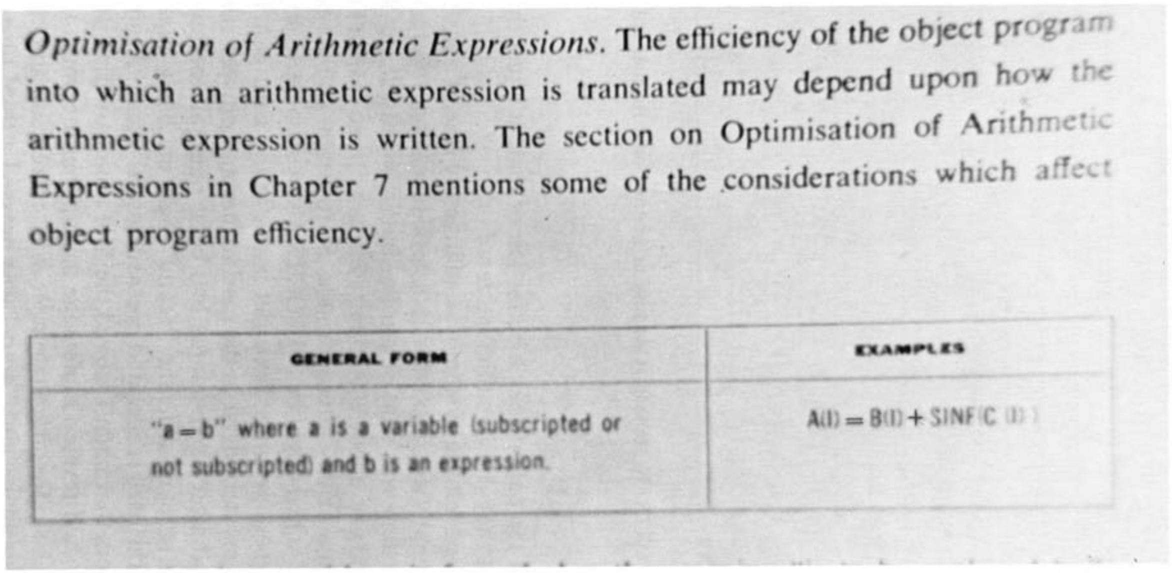


Example of a Fortran Program

The following brief program will serve to illustrate the general appearance and some of the properties of a FORTRAN program. It is shown as coded on a standard FORTRAN coding sheet.

```
C← FOR COMMENT
STATEMENT  CONTIN-
NUMBER     UATION        FORTRAN  STATEMENT                           IDENTI-
1      5   6      7                                               72  73  FICATION  80
C                 PROGRAM FOR FINDING THE LARGEST VALUE
C          X           ATTAINED BY A SET OF NUMBERS
                  BIGA = A(1)
                  DO 20  I = 2,N
                  IF (BIGA - A(I)) 10, 20, 20
      10          BIGA = A(I)
      20          CONTINUE
```

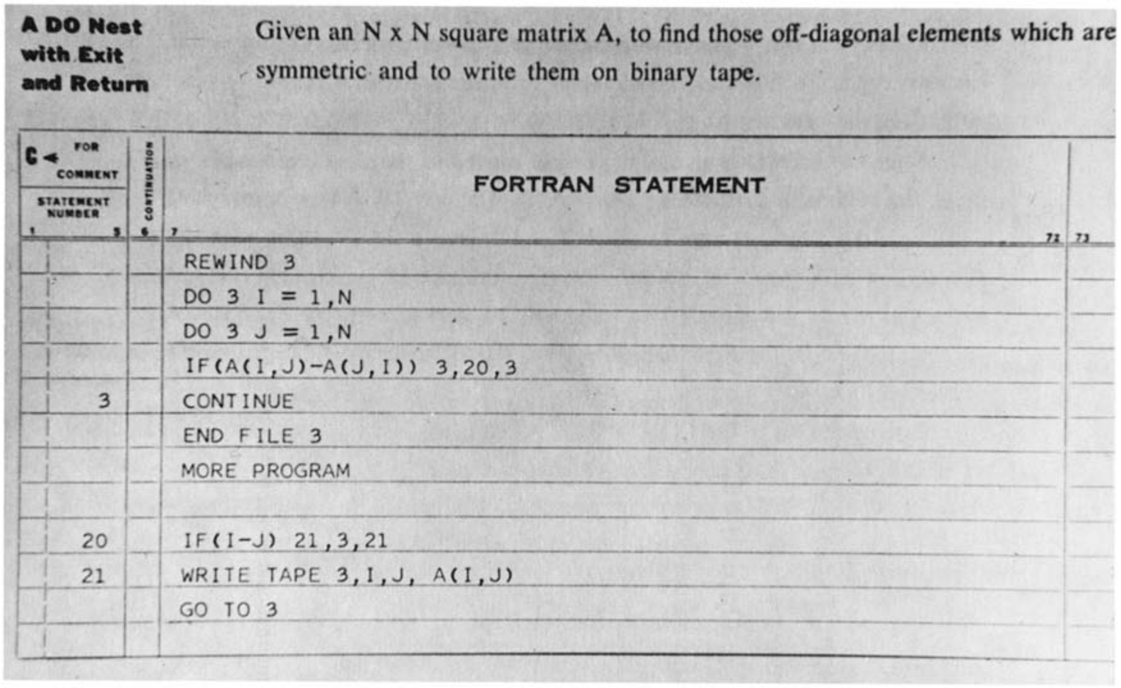
Frame 21. From the first FORTRAN Manual. [Courtesy of International Business Machines Corporation.]



Optimisation of Arithmetic Expressions. The efficiency of the object program into which an arithmetic expression is translated may depend upon how the arithmetic expression is written. The section on Optimisation of Arithmetic Expressions in Chapter 7 mentions some of the considerations which affect object program efficiency.

GENERAL FORM	EXAMPLES
"a=b" where a is a variable (subscripted or not subscripted) and b is an expression.	A(I) = B(I) + SINF(C (I))

Frame 22. From the first FORTRAN Manual. [Courtesy of International Business Machines Corporation.]


A DO Nest with Exit and Return

Given an N x N square matrix A, to find those off-diagonal elements which are symmetric and to write them on binary tape.

C FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
		REWIND 3
		DO 3 I = 1,N
		DO 3 J = 1,N
		IF(A(I,J)-A(J,I)) 3,20,3
3		CONTINUE
		END FILE 3
		MORE PROGRAM
20		IF(I-J) 21,3,21
21		WRITE TAPE 3,I,J, A(I,J)
		GO TO 3

Frame 24. From the first FORTRAN Manual. [Courtesy of International Business Machines Corporation.]

project, but I hope that the paper does a better job of that. It's just that there's a lack of time to talk about what they did in sufficient detail.

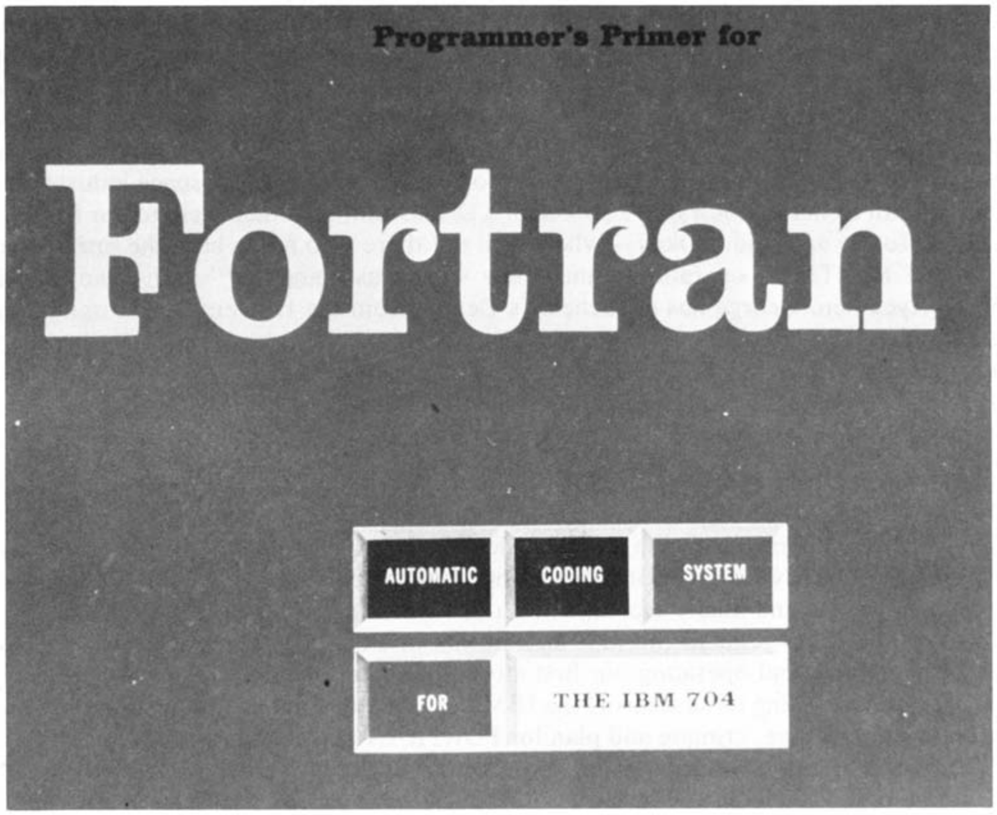
About April of 1957, we felt the system was sufficiently free of bugs to distribute it. But before the system was finally distributed on magnetic tapes there was a vast confusion and a binary deck of the system was sent out to the Bettis installation at Westinghouse; this was just an unlabeled box of about 2000 binary cards with no instructions or data of any kind. It arrived there on a Friday afternoon, and Herb Bright and his colleagues at Westinghouse suspected that it just might be the FORTRAN system deck, so they proceeded to put blank tapes on every tape unit. They put cards in the card punch, and cleared the memory. They put this mystery deck in the card reader and pressed *start*. They observed some tapes moving, and after all the cards had been read, all the tapes rewound themselves and the machine stopped with the "ready" light on. They next put a small FORTRAN test program in the card reader and pressed "start" again. Again, some tapes were seen to move, plus others that hadn't moved the first time. And this time there was some action at the printer. It had printed a message saying that a comma was missing from a particular statement. The offending statement was corrected, and the source program placed in the card reader again, and "start" was pressed a third time.

After a few seconds of computing, the card punch produced a small deck of binary cards. Bright and his friends put the binary deck in the card reader and pressed the "start" button for the fourth time. Again, there were a few seconds of computing, and then the printer started up once more. This time it produced 28 pages of results. All the output from their test program was actually correct, with, of course, the exception of a few FORMAT errors. That took place on April 20, 1957, and it was probably the last time that Westinghouse Bettis was to have such a trouble-free run of FORTRAN for a long time.

Anyway, after the system was distributed, the proprietors of the six sections were kept very busy tracking down bugs. Hal Stern collected a stream of telegrams, letters, and phone calls from all over the country and distributed them to the appropriate people. He then collected the corrections and made up correction cards that he distributed back to all the installations.

In the meantime, Grace Mitchell had been working on a programmer's primer. It came out in the winter of 1957 and a slightly revised edition came out in March of 1958 [Frame 25]. By the fall of 1958, over half the instructions for some 66 704s were being produced by FORTRAN. SHARE, the 704 user's group, had made it their second official language (the first being SAP).

There is just enough time left, I hope to let me say a little about FORTRAN II. FORTRAN II was designed mostly by Nelson, Ziller, and myself. It provided some much needed subroutine definition facilities, and the ability to compile subroutines separately. It kept their symbolic names in the binary decks for each subroutine. These independent subroutine decks could be loaded by a linking loader that would put them together properly. It came out in the spring of 1958.



Frame 25. Cover of first FORTRAN Primer. [Courtesy of International Business Machines Corporation.]

FORTRAN III was designed by Irv Ziller. It provided the ability to intermix FORTRAN statements and machine instructions with FORTRAN variables as addresses. It also introduced a number of features such as Boolean expressions and string handling facilities that later appeared in FORTRAN IV. In fact, most of the facilities of FORTRAN IV are to be found in FORTRAN III. It was mostly an internal IBM system and was first used in the winter of 1958–1959.

So, in closing, let me remark that there seems to be general agreement among the original FORTRAN group that it was one of the most enjoyable projects any of us ever worked on. We were graced by this wonderful independent status that Cuthbert Hurd and my subsequent bosses conferred on us, so that we were completely on our own. We knew we were going to provide a valuable tool, even if few others did, and we worked quite hard. But perhaps the best part was the uncertainty and excitement of waiting to see what kinds of object code all that work was finally going to produce. I only wish that programming today could be half as exciting and enjoyable as the FORTRAN project was.

Thank you.

TRANSCRIPT OF DISCUSSANT'S REMARKS

JAN LEE: Thank you, John. We invited John to prepare a paper on FORTRAN, and then we searched around to find somebody who might have some other thoughts, and I use that word "other" very carefully. We were somewhat concerned it might be "opposite" thoughts, "corroborating" thoughts, or any other. And we looked hard and long. You may notice that John said that one of the motivations for FORTRAN in those early days was this growing space industry. And so we looked to a down-to-earth space industry, and came up with General Motors, as a source of a Discussant. And then went to our friends at General Motors and said, "Okay—who was it out there who really bore the brunt of trying to get FORTRAN operational and knew what was going on?" and came up with George Ryckman. George has a Bachelor's Degree from the University of Virginia, and then went to the University of Michigan where he obtained a degree in Electrical Engineering. He's been at General Motors since 1952, starting off as a research engineer, supervisor of computer operations when FORTRAN was getting its feet on the ground, and now Assistant Department Head of the Computer Science Department at General Motors.

GEORGE RYCKMAN: Thank you, JAN. My remarks will deal primarily with the impact FORTRAN had on our organization, General Motors Research Laboratories, although I will have a few comments on the language design and its implementation. I'll begin with some pre-FORTRAN history, followed by our early experience with FORTRAN, some operational issues, and finally a comment on our more recent history.

In the three years prior to the first distribution of FORTRAN, we were preoccupied with programming and operating our first stored program machine, the IBM 701, and in preparing and installing its successor, the IBM 704. Still we took advantage of several opportunities to examine, critique and plan for FORTRAN at our installation. For example, my log book notes on March 17, 1955: "Received proposed additions to the FORTRAN: incorporated new symbols and formulas." And another entry on March 16, 1956: "FORTRAN code plates ordered for the keypunch machines."