the mean free error time on the 704 was a major factor in running large programs; it just didn't hold out that long. The economics of compiling, testing, and recompiling were typically prohibitive with programs exceeding 300–400 statements. Later with the advent of FORTRAN II, the ability to break up a program into pieces substantially removed this restriction and the language became firmly established with virtually all application programming at GM Research.

At this point I'd like to address some of the operational issues of the earliest FORTRAN, the manner in which the compiler/translator ran on the 704. It did not come with an operating system, but rather took over the whole machine when it ran, so a programmer typically approached the machine, getting four tape units ready, a printer, a card reader, and a card punch. During translation, listings appeared on the printer and a binary object deck issued forth from the punch. The programmer transferred this binary deck from the punch to the reader, pushed "GO," and hoped. He finally retrieved the printed output and the card decks, and that completed a session. Rarely could all of this activity take place in less than 10 or 15 minutes per session, so machine time was scheduled in 15-minute blocks. All of this, I should say, was in marked contrast to our normal mode of operating a machine. Earlier in 1955, in a joint effort with North American, we had designed and implemented a so-called "multijob, tape-in, tape-out monitor system" for the 704. I guess today we'd call it an operating system. The resulting throughput advantages prompted us to initiate a similar system for FORTRAN, and in fact, we had already designed and all but checked out a new system for FORTRAN at the time it arrived. Within a month it was also running in an operating system. Incidentally, after suggesting SENSE LIGHT and SENSE SWITCH additions to FORTRAN, this nonstop automatic operation was rendered almost useless.

Fortunately the earlier monitor had done a good job of tuning our computer system because the translator was highly reliant and extremely brutal on tapes and drums. From an operational standpoint the only problem that plagued us for any extended period was the numerous program halts built into the translator. But the translator eventually learned that we would not put up with such laziness, and that if it had trouble with one source program, it should simply go on and get the next one.

In the intervening years, the FORTRAN load built up to virtually 100% of our computing. Only in the last ten years has it yielded grudgingly to PL/I; until today it still accounts for 50% of the load on three IBM 168 systems.

Our best, most efficient numerical analysis codes are still written in FORTRAN, as are some huge programs for structural and fluid flow analysis, just to name a few.

I've enjoyed delving into the archives for this discussion and also sharing it with you. I wish to thank the sponsors for the invitation to join you and the privilege of participating.

TRANSCRIPT OF QUESTION AND ANSWER SESSION

JAN LEE: Thank you, George. So, John, let me start off with the most general question. It is from Nancy Stern, and really it's not directly related to FORTRAN. "I notice that in the May 1954 article, the term 'programmer' was used. From your experience, when did the term 'programmer' begin and replace the word 'coder'?"

JOHN BACKUS: Well, the shortest answer I can give to that is—it's the same reason that janitors are now called "custodians." "Programmer" was considered a higher class enterprise than "coder," and things have a tendency to move in that direction.

LEE: Aaron Finerman, currently at SUNY, Stony Brook, says, "FORTRAN, as you noted, was greeted without enthusiasm by a skeptical if not hostile audience. To what then do you attribute its rapid acceptance? Was it the ease of programming, the efficient code it produced, or was it simply the thing to do? As a skeptical manager at the time, I have my own perspective but would like to have yours."

BACKUS: That's dangerous territory. I think there are a zillion different reasons why people took it up ultimately, but mainly it was economics. That's my belief—that it was the sort of a proposition you couldn't refuse unless you had problems of a special nature that FORTRAN didn't deal with very well.

LEE: Bob Bemer from H.I.S. asks: "How about a word on how long it took Sayre and Goldberg to pick up Best's section when he left your group?"

BACKUS: That was just something I would have liked to have mentioned in my talk but I didn't feel I had time. Sheldon Best wrote this enormously complex section 5, and then rather promptly left and went back to MIT before it was completely debugged. So David Sayre and Dick Goldberg, two good scholars, studied and studied and diagrammed the section and completed its debugging.

LEE: Was that a short episode, or a long one?

BACKUS: Well, it was no longer than the whole episode of debugging. That was quite long.

LEE: Tom Steel from AT&T asks: "Would you comment on the extent to which other works, specifically PACT, influenced FORTRAN development?"

BACKUS: Well, PACT was an ambitious project instituted by a number of aerospace companies in the Los Angeles area that performed a lot of services for the programmer, but —well, to make a short answer, I believe it had no influence because it began, you know, about the time we had frozen our design, if I'm not mistaken. At least, as far as we were aware then, that was the case.

LEE: Mr. Peterson from Bonneville asks: "Concerning FORTRANSIT, what connections existed between Perlis's IT and your FORTRAN?"

BACKUS: Well, IT was an algebraic translator for the 650, and there were lots and lots of 650s around in universities. And so a translator was written—Bob Bemer is the authority on this. I think he wrote it. A translator to translate FORTRAN programs into IT programs so that you could run them on 650s.

LEE: From Mike Shapiro of NCR: "Your paper tries to dispel the legend that three subscripts came from three index registers, and yet a later FORTRAN had a seven index register machine, and therefore had seven subscripts. What are your *current* thoughts on how many subscripts are needed?"

BACKUS: Well, I stick by my original story. The complexity of analysis that Bob Nelson

and Irv Ziller had to do to optimally deal with indexing and looping rose exponentially as the number of subscripts went up. And that was, indeed, the reason we limited it to three. The fact that some later group decided to allow seven subscripts in a seven index register machine—that was out of my control. [Laughter]

LEE: Dr. Brebner from the University of Calgary asks: "With respect to FORTRAN, what do you consider was your best design decision, and what was your worst?"

BACKUS: Gosh! I just can't answer that right now. I'll try that one later.

LEE: Helen Gigley asks: "Why were the letters *I* through *N* chosen to designate integers?" [Laughter and applause]

BACKUS: Well, it just seemed for a while that people always used *I*, *J*, and *K* for subscripts, and we thought we'd be generous and add a few more.

LEE: Again from Helen Gigley: "Why was the decision to check at the bottom of a DO loop made?"—as opposed to the beginning, I presume.

BACKUS: For efficiency. You would have had an extra instruction, at least, if you'd done it otherwise.

LEE: Richard Miller asks—again, on the looping construct and the conditional construct: "Who wanted them originally, and was any support given by the hardware?"

BACKUS: No support was given by the hardware, to my knowledge, except that it provided the usual instructions that made it possible. And as far as who wanted it—we didn't know the answer to that. We just provided it, because it seemed like a convenient thing to do.

LEE: From David Dahm from Burroughs—and I think this goes back to Grace's talk. In her three-address system, pointing out that in Grace's system it was $x + y = z$, why did you decide to put the variable on the left of the equal sign as opposed to the right of the equal sign?

BACKUS: We thought it made programs more readable. If you put them on the right, the variable being assigned to would be a little hard to find, whereas, they're easy to find if they're on the left.

LEE: Juris Reinfelds from the University of Wollongong, Australia, asks: "Why and when was the decision made to interpret FORMATs, and does this not contradict your assumption of developing efficient execution code?"

BACKUS: I think Roy Nutt should answer that question.

ROY NUTT: That was strictly a matter of expediency. The original intent was to compile FORMAT statements, but we just couldn't get it done in time.

LEE: Roger Mills from TRW asks: "Was it your plan to have a subroutine facility from the very beginning, or did FORTRAN I show you it was a problem?"

BACKUS: Well, as the preliminary report indicated, we did intend to have it from the beginning, and in fact there were lots of things that never got into FORTRAN that were difficult to do—for example, summation never got in.

LEE: From Mike Shapiro again: "What comment do you have on the current trends in FORTRAN extensions?"

BACKUS: Well, I have some remarks on that in my paper. Not about FORTRAN extensions in particular, but about programming languages in general. I'm not in favor of any of them. I think conventional languages are for the birds. They're really low level languages. They're just extensions of the von Neumann computer, and they keep our noses pressed in the dirt of dealing with individual words and computing addresses, and doing all kinds of silly things like that, things that we've picked up from programming for computers; we've built them into programming languages; we've built them into FORTRAN; we've built them into PL/I; we've built them into almost every language. The only languages that broke free from that are LISP and APL, and in my opinion they haven't gone far enough.

LEE: John, thank you very much.

FULL TEXT OF ALL QUESTIONS SUBMITTED

BOB BEMER

How about a word on how long it took Sayre and Goldberg to pick up Best's section when he left?

M. A. BREBNER

With respect to FORTRAN, what do you consider was the best design decision, and what do you feel was the worst design decision?

DAVE DAHM

Why did you decide to put the variable being assigned on the left rather than the right of the "=" sign?

Where did the idea of the subscripted variable come from originally?

BILL DERBY

What language was used for your FORTRAN compiler, and how easily did it translate to the 704's successor?

KEN DICKEY

How were the bounds of your Monte Carlo simulation flow analysis (section 4 of compiler) arrived at (i.e., how did you bound your system)?

HAROLD ENGELSOHN

In the initial stages of planning, was there any conscious effort to allow for data processing as well as algebraic manipulation? If not, when were alphabetization capabilities, for example, added to FORTRAN?