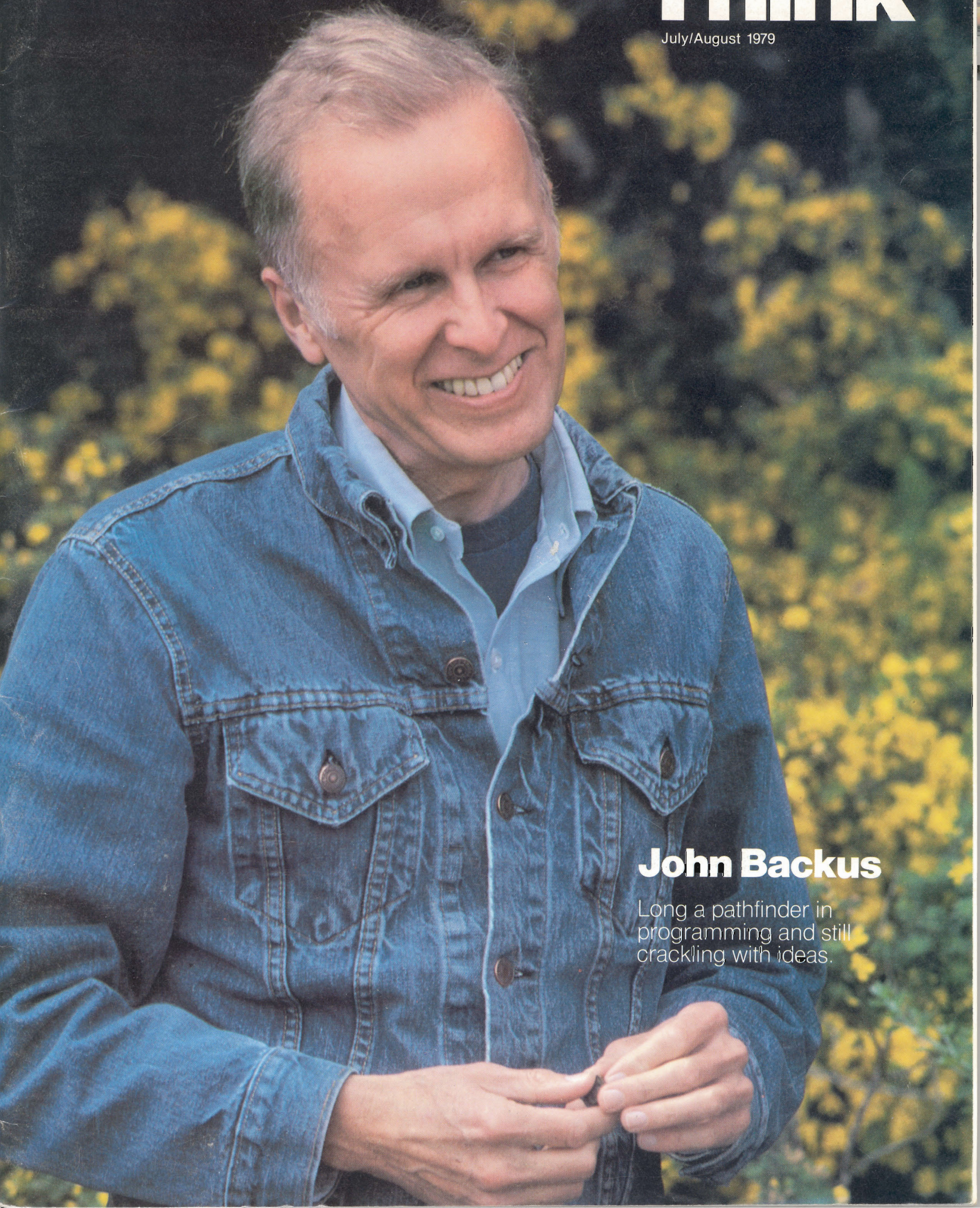
Think

July/August 1979



John Backus

Long a pathfinder in programming and still crackling with ideas.

They were an eager, absorbed young group that spring of 1954: three IBM computer programmers and a former U.S. Foreign Service employee hired to do technical typing. Their offices were tucked away on the 19th floor of the annex to what was then known as IBM World Headquarters—down the block from Tiffany's on the busy corner of Manhattan's 57th Street and Madison Avenue. Far below, in the ground-floor display center, was the machine they were trying to improve upon. It was the IBM 701 computer, which, only the year before, had launched the company into a brand new world of electronic data processing.

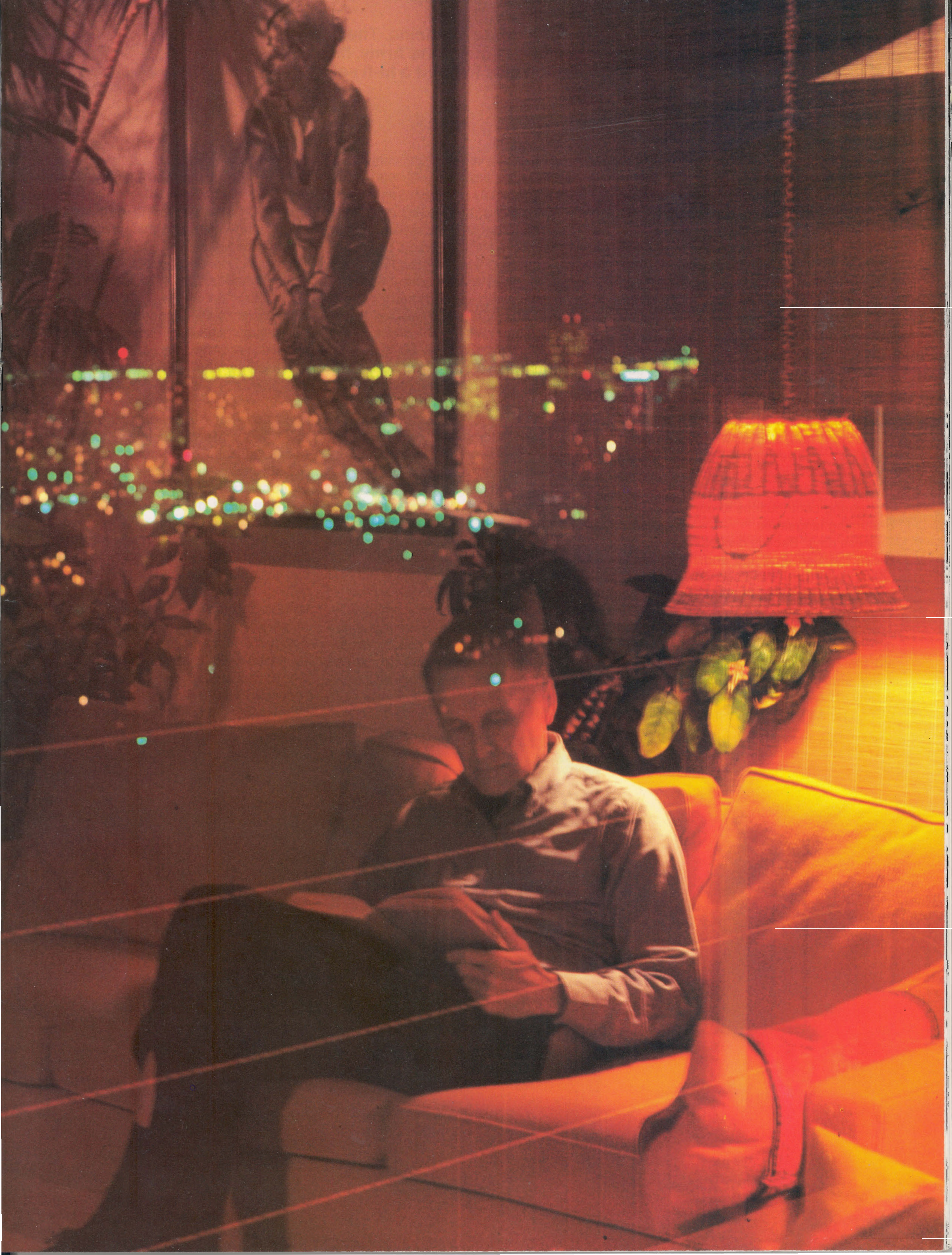
By November, they were ready with a preliminary report. Based on what the group's manager, John Backus, then 29, calls "more faith than knowledge," it stated that the programming language they had designed for the new 704 would enable it "to accept a concise formulation of a problem in terms of mathematical notation and to produce automatically a high-speed 704 program" for its solution. The report suggested that the automatic program would run as fast as a program painstakingly coded by a human programmer. Months of testing would prove them right. They named the language FORTRAN, for FORmula TRANslation.

Backus, now an IBM Fellow, went on to become a staff member of the Thomas J. Watson Research Center in Yorktown Heights, N.Y. Sixteen years ago, he traded the East Coast for California and the IBM Research Laboratory in San Jose. Of the group's other members: Irving Ziller is now a special consultant to IBM Vice President and Data Systems Division President John E. Bertram. Harlan Herrick is a systems engineer for the General Systems Division in Manhattan. The typist, Robert A. Nelson, who showed a talent for FORTRAN design, is an IBM Fellow

PATHFINDER

For 25 years a programming pioneer, John Backus still crackles with ideas.

by Claire Stegmann



He considered medicine. Then, when he got out of the Army, his main ambition was to build a hi-fi set.

and an important contributor to the virtual storage concept. As for the 704, IBM's first machine with magnetic core memory has long since become a museum piece. And last year, to make way for IBM's new 43-story skyscraper now under construction, the old headquarters annex fell to the wrecker's jackhammer.

But though the FORTRAN primer is more than 20 years old, and subsequent languages have sprung forth like dandelions on an April lawn—some 165, at last count—FORTRAN has proved exceedingly durable. Adapted to subsequent computer models, it is, today, one of the most widely used computer languages in the world. Three years ago, Backus went to the White House to receive the nation's top award for scientific and engineering achievement, the National Medal of Science, for pioneering contributions to computer programming languages.

These days, Backus divides his time between the San Jose lab and his cliffside home near San Francisco's twin peaks, where he lives with his writer wife, Una Stannard, and maintains an office with a spectacular view. He answered his front doorbell one afternoon not long ago, and led me up a flight of stairs into a bright living area filled with wicker furniture, a variety of healthy green plants and abstract paintings. As we seated ourselves by the window, the city, far below, gleamed white in the afternoon sun.

Now that FORTRAN is programming history, I wondered what its author was doing for an encore. Backus jumped up quickly and disappeared into the next room, returning with a magazine reprint in his hands. "I've had an article published in *Communications*," he said, modestly neglecting to mention that it was the Turing Award address he had given before the Association for Computing Machinery, in recognition of his early technical work. "The import of what I said," he continued, "is that conventional programming languages, including FORTRAN, are very poor languages for telling computers what to do, basically because you can't say very much. The kinds of languages I'm into now are radically different. It's all very exciting for me, because this is what it was like back in the FORTRAN days. I mean, it's this completely new field, with all kinds of questions coming up."

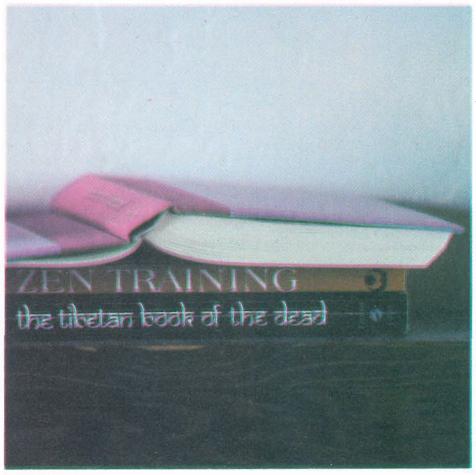
For a "pioneer," Backus is a young 54. He wears jeans as lithely as a teenager. When he talks, he gestures emphatically.

"I guess the best analogy comes from the development of mathematics," he explained. "Mathematics, you know, started with arithmetic, and then it got into slight abstractions, like simple algebra, simple equations, and then it got into questions of the structure of algebraic laws for the operations of arithmetic. What we've been stuck with in programming is analogous to the arithmetic stage. And what I'm trying to do is move from that hideously complicated manipulation of numbers up to abstractions, where you have structure and you can reduce a whole set of rules to one simple rule. If I succeed, hopefully, we'll have an intellectual foundation for a lot of new computer designs."

Surprisingly enough, for one who has his master's degree in mathematics, Backus didn't set out to be a mathematician. The son of a Wilmington, Delaware, chemist-turned-stockbroker, he had, in his words, a "checkered educational career." In and out of prep school from the 8th grade on, he spent six months at the University of Virginia, marking time until the Army draft. He had thought he might become a doctor and, once in the Army, he studied premed and began medical school at what is now New York Medical College. "I had visions," he recalls, "of right away doing research on the functions of the brain. But at medical school, all they wanted you to do was memorize, memorize, memorize. By the time I got out of the Army, my one ambition was to build a good hi-fi set."

It was at radio technician school, under the G.I. Bill, that Backus discovered math.

Backus reads widely in other disciplines. "Computers just don't speak the way people do," he says. "It would be lovely if they did."



He went on for his master's at Columbia University and, while a student, went down one day to Madison Avenue and 57th Street to have a look at what a classmate had described as "an interesting thing."

The interesting thing, it turned out, was the IBM Selective Sequence Electronic Calculator (SSEC). Installed at IBM headquarters in 1948, it was the response of company engineers to IBM's belief that electronics was the new growth area. In the early postwar years, the first electronic computer, the ENIAC, was nearing completion at the University of Pennsylvania's Moore School of Electrical Engineering; the mathematical genius of John von Neumann was bringing the stored program concept to realization at Princeton's Institute for Advanced Study; while at Harvard, IBM's own Mark I Automatic Sequence Controlled Calculator was doing multiplication and division in seconds.

Equipped with 13,000 vacuum tubes, 23,000 relays and a large number of paper tapes, the SSEC was a hundred times faster than the Mark I. The company that supplied its lubricants advertised it in *The Saturday Evening Post* as the Oracle on 57th Street. Customers used it to design turbine buckets and solve oilfield exploration problems. And Wallace J. Eckert, the director of IBM's Watson Scientific Computing Laboratory, did SSEC calculations of the moon's orbit that would show up 20 years later in the Apollo space program.

Heady stuff for a young math major. When the IBM systems service rep who put the SSEC through its paces learned that Backus would soon be in the job market and suggested he talk with the machine's co-inventor, Rex Seeber, he offered little resistance. "I had holes in the sleeves of my jacket and my shoes needed shining, but she got me an interview then and there. Seeber gave me a little homemade test and hired me on the spot."

The next two years, spent computing lunar positions, were "just delightful. You had the machine for two weeks all to yourself, just to check out your tapes and plugboards and things like that. And then, of course, you had to be there the entire time the program was running, because it would stop every three minutes, and only the people who had programmed it could see how to get it running again."

But, for Backus, the programming itself was as tedious as medical memory work. In the early 50's, most of it was being done in binary-coded numbers the computer hardware could interpret. The simplest machine instruction was a laborious process of setting down rows of "0's" and "1's" in precise order.

"Much of my work has come from being lazy," he says. "I didn't like writing programs, and so, when I was working on the IBM 701, writing programs for computing missile trajectories, I started work on a programming system to make it easier to write programs for the 701. And that wound up as something called Speedcoding."

Later, when the IBM 704 was in development up at the old Homestead lab in Poughkeepsie, Backus persuaded the designers to build directly into its hardware the features that Speedcoding simulated.

"From then on," he said, "the question became, what can we do for the poor programmer now? You see, programming and debugging were the largest parts of the computer budget, and it was easy to see that, with machines like the 704 getting faster and cheaper, the situation was going to get far worse."

Backus decided there might be a way to use mathematical notation to address the computer and have it work out its programs of "0's" and "1's" automatically. He wrote a letter to his boss, Dr. Cuthbert Hurd, head of the applied science department, saying that it might be possible to develop an automatic programming system for the 704, and make it practical. Hurd, who, in 1951, had the foresight to encourage the company to hire John von Neumann as a consultant, said yes.

Most people, Backus says today, "think

"Designing a language depends on having a couple of ideas. And where ideas come from is hard to understand. Particularly, years later."

At first, the language caught on slowly, but soon more and more scientists were doing their own programming.

FORTRAN's main contribution was to enable the programmer to write programs in algebraic formulas instead of machine language. But it isn't. What FORTRAN did primarily was to mechanize the organization of loops." A loop, heavily used in scientific work and in computing payrolls, is a series of instructions repeated a number of times until a specific result is reached.

FORTRAN did greatly increase programmer productivity. What had previously taken 1,000 machine instructions could now be written in 47 statements. And, as intended, more scientists and engineers learned to do their own programming. But the language was slow, at first, in catching on. "Users," says its creator, "just found it hard to believe that a machine could write an efficient program."

It could. By the fall of 1958, more than half the machine instructions of the 704 were being generated by FORTRAN. It was soon being used on other machines as well. "*In* a way, FORTRAN was a great boon to our competitors," says Backus, "because with their programs tied up in machine language, IBM customers weren't about to re-program for another computer. But if a competitor could come up with a program that would translate a FORTRAN program into the language of *his* machine, he had a selling point."

The telephone rang, and he crossed the room to answer it. "I've had an associate for several months now," he said, returning. "A former associate professor at Cornell. When I'm working at home, we often spend an hour a day on the phone."

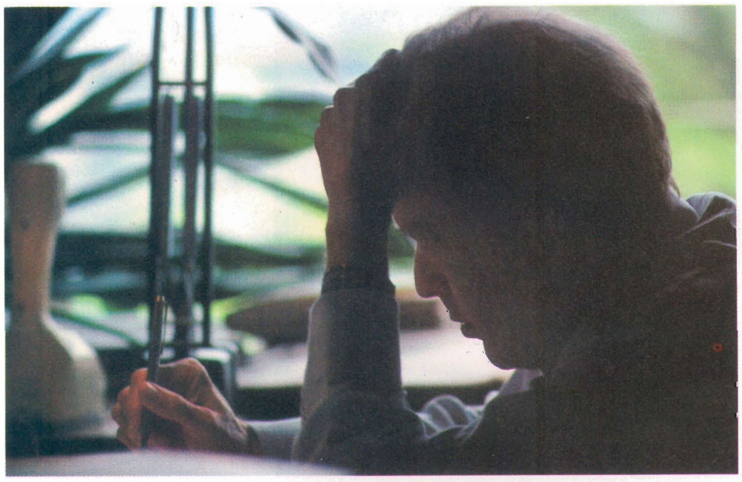
What had inspired his new work?

"I just got sick of seeing more and more new programming languages— what I call von Neumann languages," he replied. "They've just become so baroque and unwieldy that few of them make programming sufficiently cheaper to justify their cost. FORTRAN started the trend. You see," Backus continued, "all programming languages are essentially mirrors of the von Neumann computer. Each one may add a gimmick or two, to automate some of the dirty work, but it's usually done at the price of a much more complicated language. Today's programming manuals are *that* thick." He held up a thumb and forefinger. "Some of them have 500 pages. It's just a vicious circle, because language designers design to fit the computer, and computer designers think they must design to fit the languages.

"Von Neumann's concept was brilliant, of course, and worked fine 30 years ago," said Backus. "But," he paused, making arches of his hands, "here's my highly oversimplified analysis of the von Neumann computer. It consists of two boxes. One is the central processing unit, where the calculations take place, and the other is the store, or memory. Traffic between them takes place, figuratively speaking, through a narrow passage that I call the von Neumann bottleneck. Because it is just that. You see, the purpose of a program is to make a big change in the store. But how does it do it? By huffing and puffing words [a computer word is only 32 bits—"0's" and "1's"] back and forth through the tiny passage between the store and the CPU. *One word at a time.*"

The result, says Backus, is that the programmer is left with an enormous task of how to get things out of the store, combine and pump them back into the store so that the ultimate result is achieved. Everything that can be said in a conventional programming language has to be thought of in advance, making the language huge and inflexible. "And because it takes pages and pages of gobbledygook to describe how a programming language works, it's hard to prove that a given program actually does what it is supposed to. Therefore, programmers must learn not only this enormously complicated language but, to prove their programs will work, they must also learn a highly technical logical system in which to reason about them.

"Now, in the kinds of systems I'm trying to build," he explained, "you can write a program as essentially an equation, like



"I'm trying to get a basic concept of input/output. But I've wrestled with the problem for three or four years, and I still haven't got what I want."

equations in high school algebra, and the solution of that equation will be the program you want. What's more, you can prove your programs in the language of the programs themselves. The entire language can be described in one page. But," he raised a finger, "there's a catch. They're what I call applicative languages, which means that there's no concept of a stored memory at all."

But surely a computer can't do without a stored memory?

"Well, in one sense it can," said Backus. "What I want to do is to come up with a computing system that doesn't depend on a memory at all, and combine that system, in a rather loose fashion, with one that has a memory but keeps the simplicity and the algebraic properties of the memoryless system. Then, hopefully, the process of algebraically speeding up programs can be mechanized so that people can write the simplest programs and not have to care whether they are efficient or not. The computer will do the hard work. And, more than that, perhaps a lot of programs can be written simply by describing the program you want with an equation."

The sun had left the hill and, farther out, was turning the blue of the ocean to gold. "The FORTRAN language," Backus reflected, "took about nine months to devise. I've been working on *this* project since 1970, and it's still evolving. It's been difficult because it requires breaking one of the traditions of spoken English.

"For example, when we write sentences, we interpret the sentence not by the words themselves, but in terms of what they refer to. When you say, 'the cat is running,' you don't mean the word, 'cat,' you mean the animal. The same with computer programs. When you write 'x equals y plus z,' you are certainly not referring to adding the letters 'y' and 'z.' Yet my languages do just that. I call them anti-quote languages. When you use the word 'cat' in an anti-quote language, you are referring to the word, not the animal.

"Now, if you were to say that, from now on, all English sentences are to be interpreted in this new way, everybody would be terribly confused, and of course, it wouldn't make sense in the case of English. Yet, that's essentially the change I've made in programming."

There was a taxi waiting and a plane to catch.

"I wouldn't be surprised if I've boggled you," Backus said, as he saw me down the steps. "My stuff boggles computer scientists, too, at first. It's a terrible wrench in our accustomed way of thinking and just normal language usage. But what I *can* show is that if they *do* make the switch, then a lot of advantages flow from it." ■