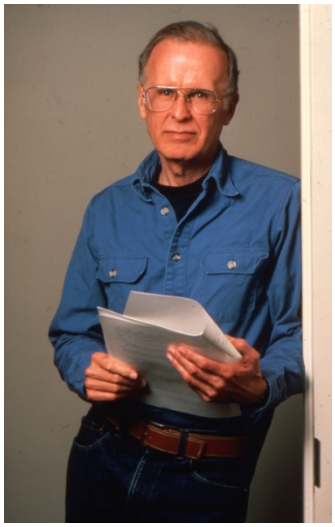
John Backus

1924-2007

A memorial given at the 2007 Conference on Programming Language Design and Implementation

Alex Aiken
Stanford University

Before I begin I would like to thank Jeanne Ferrante for giving me the opportunity to speak to you tonight, and to John Williams, Laura Haas, and the archivists at IBM for the official and personal photographs they made available to me; I've selected a few of these to share with you this evening. The suggestion for this talk came from Ron Cytron, who noted after John died in March that the popular press remembered him almost exclusively for FORTRAN, and that perhaps John would have wanted it otherwise. Indeed, in the time I knew John he had what can best be described as a love-hate relationship with FORTRAN, and for the last 15 years of his career John was devoted to functional programming. And so when I was asked to speak I thought it would be a wonderful opportunity to review the advantages of the calculus of higher-order combinator-based functional programming from the dual viewpoints of semantic foundations and program optimization . . . but then I thought the better of it. Instead, I'd like to talk first a little about John's life as a scientist and engineer, and then, a bit more personally, about the kind of life he led.



John Backus was born in 1924 and is politely described in various biographies as having been an "indifferent" or "inattentive" student. John himself was less polite and said flatly that he was a "terrible" student and a "goof-off" in high school. At his father's request he went to college and tried majoring in chemistry, but soon stopped going to classes and was expelled. The year was 1943 and he was drafted into the Army where he was first put in charge of an anti-aircraft crew, but after an aptitude test showed he had promise in medicine he was transferred to a medical training program. School still didn't agree with him, and again he lasted less than a year. Much later in life he had this to say about medical school:

"I hated it. They don't like thinking in medical school. They memorize - that's all they want you to do. You must not think."

While still in the Army and studying medicine John was diagnosed with a brain tumor; it was successfully treated, but a metal plate was installed in his head that he would carry for the rest of his life.

In 1946, John was 22 years old and out of the Army living alone in New York City. He had no idea what he wanted to do with his life; certainly nothing had worked out so far. But he did want a hi-fi set, and since in those days it wasn't easy to buy one he decided he should build one and enrolled in a class to learn radio technology. The teacher was good and took an interest in John, and John learned two important things. First, he had a natural talent for and enjoyment of mathematics. Second, you could

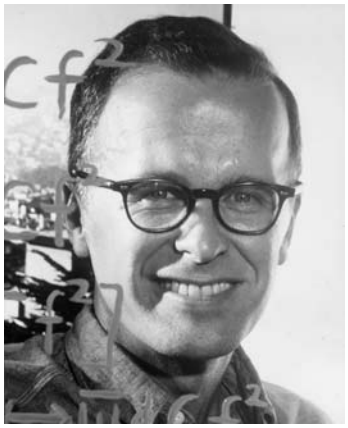
actually do useful stuff with math, like design hi-fi equipment. He went to Columbia University as a math major, and this time it worked: he graduated with a Masters Degree in 1949.

Just before he graduated he went on a tour at IBM where he saw the Selective Sequence Electronic Calculator, or SSEC. He mentioned to the tour guide that he was looking for a job, and to John's horror the guide insisted that John go straight for an interview right then and there. John was hired and spent three years working with the SSEC. One of his primary duties was to fix the machine when it broke, and the SSEC gave him a lot of practice. When it wasn't down, John was also supposed to write programs, which turned out to be a much more interesting challenge than keeping the machine running. To make programming easier, John developed his first programming language, which he called SpeedCoding. While SpeedCoding did make programming more productive, the programs ran much more slowly than programs written "by hand". Today we would say that SpeedCoding's implementation was an example of an interpreter, though at the time the term did not exist. But John had grasped the central idea that the program the programmer wrote did not have to be exactly the same as the program the computer executed. The two programs just had to be functionally equivalent.

In 1953 IBM had a new machine, the 704; Gene Amdahl and John Backus co-authored the paper describing the machine's design. IBM had high hopes for the 704, but programming had become an acute problem. The unfortunate truth was that the cost of writing and maintaining software often exceeded the cost of the hardware that IBM wanted to sell. I'd like to stop for a moment to reflect on that statement: even at the dawn of the information age, when computers cost more in both relative and absolute terms than they would ever cost again, the insubstantial, apparently infinitely malleable software was already the dominant expense. John recognized this early version of the software crisis and made a proposal to IBM's management to design and implement a high level programming language for the 704. The vision was that this language would relieve programmers of much of the burdensome detail of reducing algorithms to working code. Because the customers of the 704 were primarily scientists and mathematicians, the language would focus on allowing programmers to write their formulas in a reasonably natural notation. To solve the problem of SpeedCoding's interpreted implementation there would be a new kind of program, called the *translator*, that would literally translate the programmer's formulas into a different but equivalent 704 program that would run efficiently. Thus was the FORmula TRANslation, or FORTRAN, project born.

It took about a year to design FORTRAN, including getting extensive feedback on the new language from IBM's customers, but by 1954 there was a concrete proposal and work began in earnest on the translator, what we now of course call the compiler. John predicted the compiler would be completed in six months; instead it took two years. Again, looking back, it is striking what has changed, and what has not changed. The FORTRAN I compiler had only 25,000 machine instructions; the systems we build today are orders of magnitude more complex and sophisticated than that first of all compilers. But we are still cursed with a very limited ability to predict how long a software project will take to complete, and what unpleasant surprises lie in wait to be discovered during a project's lifetime. FORTRAN broke new ground in everything it did; it was where we learned that parsing and optimization are hard problems, for example. A member of the FORTRAN team once told a story at a dinner in John's honor that I think said quite a bit about the project and the times in which it occurred. The project was deep

into the issues of code generation and had come across a problem that no one could solve, or at least one for which they could not agree on a solution. Translation to the 704's instruction set also required selecting registers for the instructions' operands, and it wasn't at all clear how to perform this task in a way that would make good use of the registers. A meeting took place where the problem was described to John, who thought about it for a moment and then suggested that perhaps the translation should take place in two steps, first to 704 machine instructions assuming an infinite set of registers, and then subsequently a separate pass would reduce the number of registers to the number available on the machine. Apparently, then, John defined the problem of register allocation. It would be another 25 years before a widely accepted solution to the register allocation problem would be discovered, but for the FORTRAN project it was a big step forward just to recognize the problem.



After many delays FORTRAN and its compiler were released in 1957. The impact was immediate and profound. Within two years over half of all programs were being written in FORTRAN. Even considering that the entire world population of programmers at the time must have been only in the hundreds or at most low thousands, this represents astonishingly rapid acceptance of a new technology, even by today's standards. While there is no date on this photograph I infer that it is a promotional shot associated with the FORTRAN release; as you can see John is happy now that he no longer has to explain to his management why FORTRAN is still delayed.

No sooner was the FORTRAN project over than John found himself part of the ALGOL 58 and then the ALGOL 60 committees. As part of this first great effort to systematize programming language design, in 1959 John made his second major contribution, the definition of BNF, which was at first called Backus Normal Form, but later changed to Backus Naur Form in recognition of Peter Naur's improvements to Backus' original proposal. BNF was used publicly for the first time in the definition of ALGOL 60, and immediately became the standard for describing programming language syntax.

Thus, in the space of only 5 years, between 1954 and 1959, John gave us the first practical high level programming language, the first compiler, and the definitive notation for language syntax. It was the great creative period of his life, and while he did many things after this time, it is understandable that these are the contributions for which he is most remembered. As the influence of computers and computer programming spread over the next decades, people realized just how important John's achievements were, and he began to receive recognition. He was named an IBM Fellow in 1963, and received the IEEE McDowell award in 1967. President Ford awarded him the National Medal of Science in 1975 and he was elected to the National Academy of Engineering in 1977, the same year he received the ACM Turing Award. And then, in 1993, he was the recipient of the Draper Prize, a $500,000 award that spans all engineering fields for individual contributions that make the world a better place. It was only the third time the award had been given and the first time it was given to a computer scientist; the previous two awards were for the invention of the integrated circuit and the jet engine. Retired and no longer active in the field, over a decade would pass before John received what turned out to be the final honor of his life: In 2004 he was given the SIGPLAN Programming Languages Achievement Award, in,

and I quote, “recognition of a significant and lasting contribution to the field of programming languages.” Apparently we just wanted to make sure that this FORTRAN thing was going to last, or perhaps it reflects more the general ambivalence of our field towards awards, an ambivalence that, despite his many honors, John shared.

But I am getting ahead of myself and I would like to return to what John did after ALGOL and BNF. I don’t know the whole story, but I do know that John retained a strong interest in pure mathematics. For example, he worked on the four color problem for a number of years, only to see the problem eventually solved by others using, ironically, a computer program. But John’s primary passion remained programming. He grew frustrated with the limitations of imperative-style programming and felt that there must be a better, higher-level way to develop software. He eventually came to the realization that a core difficulty with FORTRAN-style languages was that programmers were reduced to reasoning about a long sequence of small state changes to understand their programs, and that a much simpler and more compositional method of reasoning would be to think in terms of the net effect of a computation, where the only thing that mattered was the mapping from function inputs to function outputs. There were others, including Robin Milner and David Turner, who were thinking along similar lines at about the same time, but as yet there was no functional programming community. That all changed with Backus’ Turing Award lecture, published in 1978. His paper, “Can Programming Be Liberated from the von Neumann Style?” did something very unusual, perhaps unique: it attacked his own previous work, his own legacy, and furthermore proposed an alternative that was far out of the mainstream. His paper had an electric effect; I think it is fair to say that it jump-started the field of functional programming. Certainly the number of researchers in the area, and the amount of funding for that research, increased dramatically in the years just after John received the Turing Award. His Turing Award lecture remains his most cited paper, and indeed remains the most cited of all the Turing Award lectures in the 40 year history of that prize.

About this time John Backus began working with John Williams, who some of you doubtless know. They are shown here together at the old IBM Research site in San Jose around 1980, both men having relocated from the East to California. Together they formed the nascent FP group, which later became the FL project, which I joined in 1988. At this point I’d like to change topics and tell you something of the John Backus I knew personally, not the John Backus of his papers and the history books. John asked to see me on my first day at IBM; he’d been out of town when I had interviewed and though I had spoken with him by phone I hadn’t actually met him yet. As I walked into his office the first thing I noticed was the large, volcano-shaped stack of unopened mail covering John’s desk. The second thing I noticed was John, happily stuffing the gigantic pile into his wastebasket, while explaining to me that he had been away for a while and so the office was a bit of a mess, but it would just take a moment for him to deal with his mail, and then we could chat. I asked him if it was all junk mail and he said he

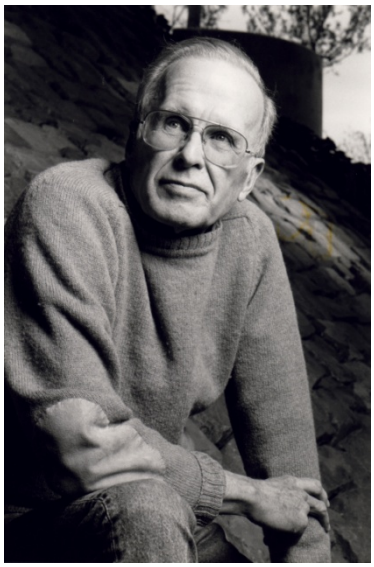
wasn't really sure, but that if there was something important they would call - they always did. I have to say that I was impressed to meet a man who had reached a station in life where he not only felt he didn't need to respond to his mail, he didn't even feel he needed to open it!

As is probably evident by now, John was an unusual person. He was very modest and self-effacing; whenever anyone sought to praise him he would deflect the praise elsewhere. At occasions in his honor he would generally respond to congratulatory comments about FORTRAN by demurring that in fact his role as the manager of the project had only been to break up the chess games after lunch so that something might get done, and that it was the members of the team who had done the real work. John always seemed to tolerate more than welcome attention to himself.

At the same time John cared deeply about ideas and especially about clear thinking. I would cringe when someone started talking nonsense in John's presence, especially if the person made the mistake of using a currently fashionable corporate motivational slogan (such as "customer driven" or "market focused"), and especially, especially if the person was a manager, because I knew what was coming next. John was almost always polite, but in his polite way he would unload on the poor soul, giving him an earful about how such sloppy logic was part of the problem with IBM and how what the company really needed was less of that kind of talk and more people who could actually do things.

Few people know that John was not only a computer scientist, but also a minister. Here he is shown officiating at a wedding, and by the way, I believe this photograph is the only existing documentation that John Backus ever wore a tie, or even knew what a tie was. Now, John was not religious, but California has very liberal laws governing religion, and any Californian can start a church that will be legally recognized by the state. Someone did just that and offered ministries in their church via an address on the back of matchboxes distributed in restaurants; on a lark John signed up and thus became a so-called "matchbox minister". But there was one consequence, which is that any minister of a legally recognized church in California can perform weddings, and so friends of John would occasionally request his services. As it happens, this particular wedding was mine.

In preparing this talk I found myself wondering how I could explain to someone who had never met John Backus and knew only his public reputation what kind of person he was. I concluded that if I could only describe one aspect of John, it would be this: That John was a rare individual who genuinely didn't worry about what other people thought of him. It is the reason he flunked out of school when he wasn't interested in it, what made him able to take a huge risk on the FORTRAN project in the face of open skepticism when he was young and unknown, and again to risk his established reputation in his Turing Award lecture when he was a senior statesman of the field he helped found. It is why he was largely



indifferent to praise, and unafraid to tell people exactly what he thought if the occasion called for it. John often said to me, and to others, that one must not be afraid to fail; that failure was important and necessary. It's advice that I must say I have followed rather more often than I care to admit, but besides being the truth, the fact that John both said it and lived it gives insight into his nature. John had fantastic successes and more than a few failures, but if he were still with us today, I believe he would say that he was content with both. Thank you very much for your attention.

The three portraits of John Backus are used with the permission of IBM; John Williams provided the photograph of himself and John Backus. Many of the facts about John Backus' early life and the quotation were obtained from various on-line sources and books.