Back to BASIC

John G. Kemeny

Thomas E. Kurtz

Creators of BASIC

The History, Corruption, and Future
of the Language

BACK TO BASIC

The History, Corruption, and Future of the Language

John G. Kemeny
and
Thomas E. Kurtz

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts · Menlo Park, California
Don Mills, Ontario · Wokingham, England
msterdam · Sydney · Singapore · Tokyo · Mexico City
Bogotá · Santiago · San Juan

Library of Congress Cataloging in Publication Data

Kemeny, John G.
Back to BASIC

Includes index.
1. Basic (Computer program language) I. Kurtz, Thomas E. II. Title. III. Title: Back to B.A.S.I.C.
QA76.73.B3K44 1985 001.64'24 84-24546
ISBN 0-201-13433-0

Set in 10 pt Century Schoolbook by Ampersand Publisher Services, Inc., Rutland, VT

First printing April 1985
ABCDEFGHIJ-HA-898765

BACK TO BASIC

The History, Corruption, and Future of the Language

INTRODUCTION

In 1963 Dartmouth College made a decision to enable all students to become computer-literate. To implement that decision required sweeping new ways to bring computing and students together. Two of those ways stand out. The first was that we needed a totally different way to distribute the computer resource; batch processing simply would not work. The second was that we needed to develop a new language, one that was far easier to learn and use for large numbers of students; none of the existing languages was suitable.

To meet these needs, the two of us, with the help of a group of exceptionally able undergraduates, developed the first fully functional general-purpose time-sharing system. We also developed a new language, one that was ideal for introducing beginners to programming and yet could serve as a language for all applications, even for large and complex software systems.

The new language was BASIC. Its wide acceptance went far beyond anything we had hoped for. That was the good news. The bad news was that its very success led to problems. There developed a proliferation of BASICs, ranging from the excellent to the terrible. If we are the parents of BASIC, there are some descendants we would frankly like to disown!

At Dartmouth we had a clear philosophy about the language. As it grew from modest beginnings to the "all-purpose" language we intended, we did not lose sight of the original principles. The name BASIC stood for Beginner's All-purpose Symbolic Instruction Code. The first word of that acronym is as relevant today as it was then. No matter how powerful the language became, we never forgot the needs of beginners.

We worked hard to make it truly general-purpose, while keeping the original clean and simple design. When computer science made the case for fully structured languages, we and our colleagues redesigned and expanded the structures in BASIC. When graphics terminals became available, we added a powerful and easy-to-use graphics package. As a result, over 90 percent of the Dartmouth students became computer-literate (that is, able to write simple programs), the computer became essential to hundreds of courses, and BASIC became the language of choice in a highly diverse academic community.

The fate of BASIC in the outside world was not so fortunate. The great improvements that were taken for granted here, and at some other institutions, did not make their way into most commercially available versions of the language. And when microcomputers appeared, first with very limited capabilities, software houses made severe compromises in order to fit BASIC into very small memories. Microcomputers are now much larger and faster even than the computer on which BASIC was born back in 1964, but the compromises have become "features," and they were kept—no matter how ugly.

Worst of all, the original philosophy that led to BASIC was lost sight of. For example, we have always believed that users should be protected from the peculiarities of the hardware, so that BASIC programs could be run on all kinds of machines. Instead, there are now several versions of BASIC on each microcomputer. It is even true that versions written by the same firm, for different computers, are grossly incompatible. And even the best of these tend to be roughly vintage 1966 BASIC.

Today the world doesn't have to be sold on the importance of computing and computer literacy. But the world has not yet fully learned what we were able to demonstrate two decades ago—that a simple, well-designed language and a simple computer interface can allow the many to know what only a few could know before.

Delivering computer power is no longer a problem—personal computers do it so much better than the crude time-sharing system we developed years ago. But computer language still remains a problem. We decided that once more we had to step in and take charge. We believe that we can construct a powerful, modern, easy-to-use BASIC for microcomputers, and still adhere to the tried and true principles we set down years ago. This new BASIC is called True BASIC. Written with the aid of three systems programmers who have a long association with Dartmouth, it will be available on most of the newer microcomputers.

Once again it will be possible to have the same program run on a variety of machines.

Over the past decades we have often been urged to recount the history of the language, why we created it, and what principles distinguish it from earlier computer languages. Although one of us (Kurtz) did write an account for experts,[1] we have never told the story to a general audience. During the intervening years other commitments made such a project impossible. Tom Kurtz served as the first director of the Kiewit Computing Center at Dartmouth and was heavily involved in regional and national computing projects. John Kemeny served as president of Dartmouth College for eleven and a half years. With the appearance of True BASIC, now seems like a good time to tell the full story.

This book is the story of BASIC, from its elementary version in 1964 to True BASIC in 1984.

The first three chapters of the book are a history of BASIC, from birth, to its growth into a powerful language, to the development of a fully structured modern language. It is a personal history, and we recount many details not previously published. We have tried to recall why we did what we did, what decisions turned out to be fortunate, and where we erred.

Chapter 4 discusses what went wrong. Unfortunately, a great deal did. If the reader cannot resist a quick peek at that chapter, we promise not tell anyone!

Chapter 5 is aimed at those millions who have programmed in some version of BASIC but have never seen the kind of BASIC we have used at Dartmouth for nearly a decade. It introduces the reader to structured programming through carefully chosen examples. We make the case that the transition from a sloppy version to the fully structured one is not difficult. And once one makes the transition, one will never revert to old habits. We know, because we had to go through this transition ourselves. The result is that long programs are much easier to write, to debug, and to read.

In chapter 6 we attempt a factual comparison with some of the more popular computer languages. Our aim is to show that structured BASIC deserves to be ranked as an equal among other "serious" languages. It is still one of the easiest languages to learn, and yet it has all the features one would want for introductory courses in programming or computer science, or for large application packages.

Chapter 7 is there purely for the fun of it. We hope by then to have whetted the reader's appetite for structured BASIC, and want to share a variety of examples. The examples are, we trust, interesting in themselves. And they illustrate the ease with which a wide variety of applications can be programmed.

JOHN G. KEMENY

THOMAS E. KURTZ

September 10, 1984

BACK TO BASIC

The History, Corruption, and Future of the Language

CONTENTS

1 THE BIRTH OF BASIC

Our memories of the birth of BASIC are still vivid after more than two decades. On most points our recollections agree, and we will share these with you. In a few cases our memories do not agree or the recollections are very personal. In these cases each will speak for himself.

THE BAD OLD DAYS

Let us recount what the normal use of computers was in 1963. Computers in the modern sense had existed since the early fifties, but they were large beasts and very expensive. Computer center directors considered it their major task to protect machines from human inefficiency. Since computers were so fast and so expensive, while human beings were slow (and cheap?), schedules were designed to maximize the amount of computing carried out by a machine.

The human user punched a program on IBM cards and submitted it to an operator. The operator collected hundreds of such jobs and fed them to the computer in a "batch." The machine worked on one task at a time, until it was completed, and then printed results on a not-very-fast printer. When all the jobs in the batch were completed, it was time for the next feeding of the computer.

The user returned the next day to pick up the results. Since the computer could carry out in seconds what would take a human being weeks or months, it seemed reasonable to wait twenty-four hours for the results. And so "batch processing" was universally accepted.

It might indeed have been a good system if the user received a solution the next day. But what he or she found on the printout was something like "Illegal instruction on card 27." And then the long process of finding errors, or "debugging," began. The first few days were needed just to correct keypunching errors.

KEMENY: One of the happiest days in my life was when I realized that I would never have to punch another card! The keypunch was an awkward device, and any punching error required tearing up the card, and starting over, since there was no reasonable way of plugging up the little holes.

Then one day there was not even an error message, just a blank piece of paper. Perhaps the computer had calculated the answer to the problem, but the user forgot to specify that the answer should be printed, so it wasn't! An additional card was hastily inserted—too hastily—and the result was twenty printed pages jammed full of numbers, when you expected a one-line answer. And so it continued until, about two or three weeks later, you actually obtained a solution to your problem.

KEMENY: I first became aware of the shortcomings of batch processing in the summer of 1956, as a consultant to the RAND Corporation (Santa Monica, California). I saw highly paid experts stand in line for hours to get a five-second debugging shot. I left behind a recommendation that a system be devised to allow brief debugging interruptions to the batch system. But I had not yet faced the need to change the system of computer use completely.

KURTZ: I remember one of my early experiences at Massachusetts Institute of Technology, in 1956 or '57, with batch processing. I was then a poorly paid instructor and research associate, so it was of no consequence that it took several months to solve my problem (I visited MIT once every two weeks, by train). But it was a shock to realize that I had used up more than an hour's worth of very valuable computer time on the IBM 704! It must have been all those "memory dumps" I took to study between visits to MIT.

As we began to plan a computer system for Dartmouth, it became clear that these old ways would not work. Scientists and engineers might be willing to put up with such service, but we were convinced that Dartmouth students would rebel.

A BETTER IDEA

There were some experiments going on (notably at MIT and Bell Labs) that would allow a number of people to use the same computer at the same time. The great speed of the computer would allow all of them to receive fast service. And although people would make just as many mistakes, error messages would appear within seconds, and one could get a program debugged in fifteen minutes to half an hour. It was expected that the system would be less efficient but much kinder to human beings. The machine waited on human convenience, not the reverse! And the ultimate irony would be that these user-friendly systems would prove *more* efficient than batch processing. As Kurtz recalled, with batch processing users tended to ask for enormous and costly "memory dumps" that were totally unnecessary if one received fast response and could try a correction immediately.

KURTZ: We started thinking about time-sharing in the early sixties. While I was visiting John McCarthy at MIT, he told me, "You guys ought to do time-sharing." I went back to Dartmouth, said the same thing to John Kemeny, and he instantly agreed. There simply was no question in either of our minds.

KEMENY: It was in 1962, while I was chairman of the Dartmouth Mathematics Department, that my colleague Tom Kurtz came to me with an outrageous suggestion. He started by asking, "Don't you think the time is approaching when every student should learn how to use a computer?" And I said, "Sure, Tom, but it isn't physically possible to teach so many students." And then came Tom's radical proposal: "I think we could design a completely different way of using computers that would make it possible to give computer instruction to hundreds of students."

At the time, Dartmouth had only a very small computer. We had already learned that able undergraduates could achieve incredible results with the most primitive computer. But the computer was so small that it could be used by only one student at a time. It was far less powerful and had far less memory than today's personal computers. It therefore had to have quite fancy software if many students were to use it. So we began experimenting.

Kurtz: In one experiment with this computer, which was an LGP-30, we were able to allow about five students to complete their small programs in a total time of fifteen minutes. Each student could have two or three "turnarounds" in that fifteen-minute period. Then another five students would show up. This convinced me that an easy-to-use interactive system could allow hundreds of students to use it and thus learn about computers.

Kemeny: Tom's suggestion—to open computing to all students—was a radical one, way ahead of its time. It would significantly alter the modern history of the college and have a major national impact.

We decided to design and implement such a time-sharing system at Dartmouth. We came up with a totally new computer architecture, in which one computer took care of all communications and scheduling, while the other could concentrate on serving one user at a time. We used two General Electric (GE) computers for tasks their designers never intended, and set Dartmouth on a road that would put it into a pioneering role in computer education.

We were very fortunate to receive full support from President John Sloan Dickey and the Board of Trustees. Dean Leonard M. Rieser, a physicist, was an early and ardent supporter. And the dean of our engineering school was Myron Tribus, one of the most far-sighted people we know. He became a great booster. We believe that they, with the help of a key trustee, were responsible for getting full Dartmouth backing. We were even allowed to "go GE" when most of the world was pressuring us to "go IBM."

We approached the National Science Foundation (NSF) for the necessary funds (mostly for equipment, and modest for those days). They submitted our application to outside referees. A typical reviewer said that we had no idea how large and difficult a task we were undertaking. We were hopelessly understaffed—two faculty members part-time plus a dozen students. And not even graduate students but undergraduates! To the credit of NSF, they took a chance on us in spite of the negative evaluations. And one year from the start of our project (seven months after the equipment arrived), we were teaching hundreds of students on the Dartmouth Time Sharing System.

The reviewers were right that we didn't know how difficult a job we were undertaking. If we had, we might never have tried. But they were completely wrong about the use of undergraduates. I would later say that we were first to succeed because the others used professionals while we used undergraduates! Undergraduates will work endless hours, are open to new ideas, creative, and willing to take on impossible tasks.

For example, John McGeachie and Mike Busch solved the problem of communication between two completely different GE computers, one of which did not even have a manual since it was never intended to be programmed outside the factory. It was the strangest collaboration we have ever witnessed: two students from very different backgrounds, of different temperament and appearance, each thoroughly identified with "his" computer. They would stand by their machines and yell at the tops of their voices: "Mike, you are not responding!" "Why, you never got through to me!" They may have been the original odd couple, but what they achieved, in the short time they had, was truly astounding.

It was highly appropriate that when the Federation of Information Processing Societies awarded us their first Pioneers' Day award, our students received recognition as full partners.

The coming of time-sharing fundamentally changed the use of computers. Although at the time it would have been prohibitively expensive to give each user his or her own computer, we were able to create the illusion that the large computer was just sitting there waiting to serve. It was the beginning of personalized, interactive computing. The recent arrival of personal computers will carry the availability of individualized computer service to a much larger group of users. But the essential breakthrough occurred in 1964.

KEMENY: In the midst of planning time-sharing, I made a suggestion: "Tom, if we design this great new system, why don't we also create a really nice language? Surely we can do better than FORTRAN for teaching purposes?" It led to one of the few disagreements I had with Tom. He agreed in principle that we could design a much nicer language, but felt that it might be possible to come up with reasonable subsets of FORTRAN or ALGOL. His worry was that we would teach our students a language that could not be used outside of Dartmouth.

Tom deserves all the credit for proposing the time-sharing project. But I do occasionally remind him of the fears he once had about our new language. The language is BASIC, and it is now the most widely used computer language in the world.

THE ORIGIN OF THE SPECIES

A computer is created speaking one language, called—not surprisingly—"machine language." Its alphabet consists of zeros and ones, corresponding to electronic or magnetic devices being "off" or "on." To make it slightly more palatable, the manufacturer provides an "assembly language," which allows mnemonic codes for frequently used combinations of zeros and ones, and provides certain other shortcuts. Yet assembly language is difficult to learn and use, and most computer scientists agree that one should go to almost any extreme to avoid having to code in assembly language. It is time-consuming, frustrating, hard to read, and very hard to debug.

The appearance of FORTRAN (around 1957) was a major advance in computer programming. It allowed one to write programs in a combination of a few English words and mathematical formulas. The language was designed for numerical applications, particularly for engineers. A much improved version of the language is still widely used today.

But it was intended to be used by experts. It was common to take two months of training in FORTRAN before you tried writing your first program.And the alternatives were no better for our purpose. We wanted a language for a lay audience, one that could be learned easily and whose advanced features could be learned later, when the student was ready for more sophisticated concepts.

The genealogy of BASIC really goes back to 1956. We began computing on the IBM-704, first at General Electric in Lynn, Massachusetts, and later at MIT. FORTRAN didn't arrive until the following year, so we had to learn SAP (Symbolic Assembly Program), which was assembly language. But would our colleagues be willing to learn SAP if they wanted to do computing? We thought not, and so devised a simplified version of it that we called Darsimco, for Dartmouth Simplified Code.

The idea was to provide templates, or sequences of instructions, for performing most of the simple tasks. For instance, if "A = B + C" was wanted, Darsimco said to use

```
FLD B
FAD C
FST A
```

Persons familiar with assembly language will recognize the sequence as "floating load, floating add, floating store." Our colleagues didn't have to understand what FLD, FAD, or FST meant; all they had to do was to use them as we prescribed.

Darsimco didn't catch on and therefore must be judged a failure. Besides, FORTRAN became available the following year. But the point is that we were both deeply concerned about making computing as simple as possible. Seven years later, in 1963, we finally hit on the way to do it—interactive computing with BASIC.

KURTZ: The experiment I mentioned earlier took place on the LGP-30. Steve Garland and Bob Hargraves, together with classmates Anthony Knapp and Jorge Llacer, had built a compiler for the ALGOL language. Later they constructed a "load and go" version of ALGOL. We called this system Scalp, which stood for Self Contained ALGOL Processor. All the student user had to do was to load his paper tape, and Scalp would either run the program or produce error messages. There was even a crude interactive debugger, so the student didn't have to leave the machine to make trivial corrections.

KEMENY: We experimented both with implementation of standard languages and with the creation of new, easy-to-learn languages. Our students Steve Garland and Bob Hargraves showed us what incredible achievements we might expect from undergraduates. (They would both return to