# XI

# BASIC Session

*Chairman: Thomas Cheatham*
*Speaker: Thomas E. Kurtz*

## PAPER: BASIC

*Thomas E. Kurtz*

Darthmouth College

### 1. Background

#### 1.1. Dartmouth College

Dartmouth College is a small university dating from 1769, and dedicated "for the education and instruction of Youth of the Indian Tribes in this Land in reading, writing and all parts of learning . . . and also of English Youth and any others" (Wheelock, 1769). The undergraduate student body (now nearly 4000) outnumbers all graduate students by more than 5 to 1, and majors predominantly in the Social Sciences and the Humanities (over 75%).

In 1940 a milestone event, not well remembered until recently (Loveday, 1977), took place at Dartmouth. Dr. George Stibitz of the Bell Telephone Laboratories demonstrated publicly for the first time, at the annual meeting of the American Mathematical Society, the remote use of a computer over a communications line. The computer was a relay calculator designed to carry out arithmetic on complex numbers. The terminal was a Model 26 Teletype.

Another milestone event occurred in the summer of 1956 when John McCarthy organized at Dartmouth a summer research project on "artificial intelligence" (the first known use of this phrase). The computer scientists attending decided a new language was needed; thus was born LISP. [See the paper by McCarthy, pp. 173–185 in this volume. Ed.]

#### 1.2. Dartmouth Comes to Computing

After several brief encounters, Dartmouth's liaison with computing became permanent and continuing in 1956 through the New England Regional Computer Center at MIT,



ISBN 0-12-745040-8

sporting a new IBM 704 and heavily subsidized by IBM. The earliest use of this machine was without the aid of either an operating system or a higher level language. John Kemeny and I learned the Share Assembly Language (SAP), how to read memory dumps, and the joys of two-week turnaround (my schedule of train trips to Boston). Before the IBM 704 was installed at MIT, we used a similar machine through the courtesy of General Electric at Lynn, Massachusetts.

Disturbed by the intricacy of programming in assembly language and despairing of ever teaching it to our students and colleagues, John Kemeny devised DARSIMCO (DARtmouth SIMplified COde). In it, the common arithmetic operations were expressed as templates of two or three SAP instructions. Thus, to perform `A + B = C`, one wrote as a single template:

```
LDA A
FAD B
STO C
```

Similar templates were devised for subtraction, multiplication, division, and simple looping. This was, then, Dartmouth's first crack at a simple computer language. Little used, since FORTRAN became available the following year, DARSIMCO reflected Dartmouth's continuing concern for simplifying the computing process and bringing computing to a wider audience (Dartmouth, 1956).

FORTRAN appeared on the MIT machine in 1957. Its acceptance was tempered by its alleged inefficiency in comparison with assembly language. This allegation had little effect on us, and we quickly dropped both SAP and its dialect DARSIMCO. One incident in particular comes to mind. I had a project to calculate certain statistical percentage points over an array of third and fourth moment values. Indoctrinated by the maxim that FORTRAN was inefficient, and that one could and should save valuable machine cycles by coding directly in assembly language, I did just that. After several months of trying, reading memory dumps, and trying again, I admitted failure. I had consumed about one hour of valuable 704 time, and untold hours of my less valuable time. I then programmed the same computation in FORTRAN, inefficiently, I am sure. The answers appeared. About five minutes of computer time were used. This lesson—that programming in higher level languages could save computer time as well as person time—impressed me deeply.

### 1.3. Computing Comes to Dartmouth

An LGP-30 small computer arrived at Dartmouth in June of 1959. This computer had a memory of 4K 30-bit words on a drum that rotated 30 times a second, and had 16 simple instructions. Several undergraduates were persuaded that summer to explore what could be done with the new toy. One student, without any prior background in computing, prepared a simple higher level language and language processor he called DART (Hargraves, 1959). Obviously influenced by FORTRAN, but wishing to avoid scanning general arithmetic expressions, he required parentheses around all binary operators and their operands. Hardly earth-shaking, but one conclusion was inescapable: a good undergraduate student could achieve what at that time was a professional-level accomplishment, namely, the design and writing of a compiler. This observation was not overlooked.

At the same time E. T. Irons, then a graduate student at Princeton, spent the summer in Hanover while working on a syntax-directed compiler (Irons, 1961). Our attention was

thus drawn to the International Algebraic Language known also as ALGOL 58. A group of four students (Stephen Garland, Robert Hargraves, Anthony Knapp, and Jorge Llacer) and I began to design and construct an ALGOL 58 compiler for the LGP-30. A paper (Samelson and Bauer, 1960) discussing what is now known as bottom-up parsing, provided the efficient scanning algorithm needed for a slow and small computer. Shortly, ALGOL 60 appeared (Naur, 1960), and our effort shifted to it.

Since the limited size of the LGP-30 precluded a full implementation of ALGOL 60, certain of its features (arrays called by value, own arrays, strings, variable array bounds, and recursion) were omitted; but we did include parameters called by name, using "thunks" (Ingerman, 1961; Irons and Feurzeig, 1961), and integer labels. We dubbed our work ALGOL 30, since it was for the LGP-30 (Kurtz, 1962a, Feb., 1962b, Mar.). From this project emerged a small group of undergraduate students who were well equipped to perform further work in the development of computer languages. For instance, one student (Garland) discovered that compound statements and blocks could be included in the Samelson and Bauer scanning algorithm. This simple fact was not published until some years later. (I have been unable to identify the source I clearly remember; the closest is Gries, 1968.)

The ALGOL 30 system suffered one defect that hindered its wide use as a student-oriented language: it was a two-pass system. The intermediate code was similar to relocatable binary, but had to be punched onto paper tape. Compilations could be "batched," but the delays between presenting the source code tape and the final execution were too great to allow widespread student use. It was clear that a "load-and-go" system was needed. Thus was born SCALP, a Self Contained ALgol Processor (Kurtz, 1962c, Oct.).

The design of SCALP (by undergraduates Garland and Knapp, and myself) was limited by having only one-third of the available 4K word memory for the compiler, another third for the run-time support routines (including software floating point routines), and the final third for the compiled user code. Several drastic decisions were made, the most severe of which was to limit the *for-loop* construct to only the *while* element, since one could obtain the *step-until* element as a special case.

Two other details of SCALP are worth mentioning. The student's program was compiled almost as fast as the source code paper tape could be read in. Furthermore, once in execution, a program could be traced and patched using the symbols of the source code. These two features allowed five or six student jobs to be completed in 15 minutes. Thus, quick turnaround and the ability to deal only with the source language were confirmed as essential for a student-oriented system. Hundreds of students over the next several years learned and used SCALP until it was replaced by BASIC (Kurtz, 1963a, Jan.).

Another experiment in language design occurred around 1962. The language was called DOPE—Dartmouth Oversimplified Programming Experiment (Kemeny, 1962). Kemeny was assisted by student Sidney Marshall. Intended to offer a smooth transition from flowcharting to computer programming on the LGP-30, it expressed typical flowchart operations as two or three operand commands. For example, the assignment `C = A + B` appeared as

```
7 + A B C
```

and `Z = SIN(X)` as

```
10 SIN X Z
```

Only one arithmetic operation per line was allowed. Variables could be single letters or letter–digit combinations, a harbinger of BASIC. The letters E through H were reserved for vectors, each allowing components numbered 1 through 16. In addition to the four main arithmetic operations (+, −, ·, /), the commands included four functions (SQR, EXP, LOG, SIN), a jump (T), and a three-way branch (C)—shades of FORTRAN! Loops began with Z and ended with E, and allowed only unit step sizes. Input–output commands included an input (J), input and print a label (A), print a number (P), and print a new line (N).

Though not a success in itself, DOPE presaged BASIC. DOPE provided default vectors, default printing formats, and general input formats. Line numbers doubled as jump targets. Uppercase letters and lowercase letters could be used interchangeably.

While no one of these language projects (DARSIMCO, DART, ALGOL 30, SCALP, DOPE) was an unqualified success in its own right, they all were influenced by our goal to simplify the user interface. They taught us something about language design. And they confirmed that programming talent was available as we prepared for our next computing venture.

## 2. Rationale for BASIC

### 2.1. Goals of the Project

Dartmouth students are interested mainly in subjects outside the sciences—only about 25% major in science or engineering. While science students will learn computing naturally and well, the nonscience group produces most of the decision makers of business and government. We wondered, "How can sensible decisions about computing and its use be made by persons essentially ignorant of it?" This question begged the conclusion that nonscience students should be taught computing. The hard question was thus not "whether" but "how." It seemed to us inescapable that to learn about computing one must deal with it directly; that is, one must actually write programs. Merely lecturing about computing would not work. But having large numbers of nonscience students learn to write programs in assembly language, or even FORTRAN or ALGOL, was clearly out of the question. The majority would balk at the seemingly pointless detail. What could we do?

The approaches common at the time seemed inadequate for our purpose. Typical access to computing, requiring punched cards and involving both programming and administrative hurdles, might be acceptable to technically minded students, but certainly not to others. Typical introductory courses that were really baby courses in numerical methods would not be elected by the majority of the students in whom we were interested. Furthermore, Dartmouth's English, foreign language, and distributive (humanities, sciences, social sciences) requirements weighed heavily against introducing whole new courses about computing.

We gradually settled on a fourfold approach. First, we would have to devise a computer system that would be friendly, easy to learn and use, and not require students to go out of their way. While we felt that our students would not accept punched cards, we did feel they would accept time sharing, a technique that had been demonstrated previously at MIT and elsewhere (Corbato *et al.*, 1962; McCarthy *et al.*, 1963). (Around 1960 or 1961,

after a visit to the PDP-1 time shared computer at MIT, I can clearly recall John McCarthy saying, "Why don't you guys do time sharing?" Shortly afterward I said to Kemeny, "I think we ought to do time sharing." Kemeny responded, "OK." And that was that!) Computing could thus be brought to the comfort of work areas readily available to students. The newly developed ASCII code and inexpensive communications and terminals (Teletype models 33 and 35) came along in the nick of time. And we concluded, bravely perhaps, that we could design and construct a time-sharing system that would be friendly and easy to use. We had absolutely no doubt about the easy-to-use part. As to the design and construction, we were willing to believe that our undergraduate students could tackle literally any programming job we gave them. (For many years, this proved true. Only when the systems jobs required continuous attention for more than several terms did this approach falter. Still, Dartmouth students over the years accomplished far more than anyone might have been willing either to predict in advance, or to believe by hindsight.)

The second part of the project was to develop a new language—again, one that was easy to learn and use. While Kemeny instantly realized (in the spring of 1963) the need for designing a new language, I preferred to believe that, although existing languages such as FORTRAN and ALGOL were not suitable, reasonable subsets of these languages could be devised. This belief proved wrong. For example, the compound statement is a central concept in ALGOL; one cannot construct a nontrivial *for-statement* without it. But the idea of a compound statement posed pedagogical problems; it would have to be defined and explained. I had even toyed with the notion (Kurtz, 1963b, May) that

```
for i := 1 step 1 until n do
begin a := b end next i;
```

which is legal ALGOL, could be written

```
for i := 1 step 1 until n do
a := b next i;
```

With FORTRAN, a subset wouldn't make sense if it violated the IJKLMN convention. But how many nonscience students could appreciate the distinction betwen integer and non-integer variables. Other problems included remembering the punctuation in the DO and IF statements, and the order of the three statement numbers in the IF statement. Very early, then, I agreed with Kemeny that a new language was needed to meet our requirements.

The third component of the project concerned the student's initial introduction to computing. For reasons given earlier, a whole course was not practical. On the other hand, the principal motivation for a student to learn computing would come from applications in which the student might have a direct interest. Accordingly, we made plans to introduce computing as an *adjunct* to two courses: second-term calculus, and finite mathematics. The applications of computing came from the topics of the course. Over the years about 85% of all Dartmouth students have taken at least one of these courses (Kemeny and Kurtz, 1968).

The fourth and final element was the decision to operate the computer with open access. Dartmouth has a large open-stack library, and it was natural for us to carry over this philosophy into our computing operation. Achieving open access was aided by the virtual absence, in those days, of government-supported research at Dartmouth. We were thus

freed from the major constraint found at other large institutions—the apparent requirement to charge students and faculty for computer services lest income from government grants be jeopardized.

The ideas neatly summarized above did not arise overnight, but actually evolved gradually over the years since 1956, and depended in crucial ways on the nature of Dartmouth as a small liberal arts university. Although Kemeny and I, and our student programmers, did most of the work, we were backed at all times by President John S. Dickey, Dean of the Faculty Leonard M. Rieser, and Dean of the Thayer School of Engineering Myron Tribus. In fact, Tribus writes (letter, 1977 November 7):

> I remember quite vividly an afternoon conversation with John Kemeny in which we agreed that certain requirements would be met even if it meant indefinitely postponing the computer.
>
> a. Students would have free access.
> b. There would be complete privacy. No one would know what the students were doing.
> c. The system should be easy to learn. The computer should contain its own instructions.
> d. The system would be designed to save the time of the user even if it appeared that the computer was being "wasted."
> e. Turnaround time should be sufficiently rapid that students could use the system for homework.
> f. The system would be pleasant and friendly.

In early 1964, with the assistance of two grants from the NSF and educational discounts from General Electric, Dartmouth acquired a GE-225 computer (replaced in the summer of 1964 by a GE-235) combined with a Datanet-30 and a disk that could be accessed from either machine (Kemeny and Kurtz, 1968). Kemeny had begun work during the summer of 1963 on a compiler for a draft version of BASIC; GE provided access to GE-225 machines in the area. Design and coding for the operating system began in the fall of 1963, with the main responsibility falling to students Michael Busch and John McGeachie. The pace quickened in February of 1964 when the equipment arrived on the campus. The first BASIC program under time sharing was run on May 1, 1964, around 4 A.M. Three terminals in May became eleven by June, and twenty in the fall.

The design of BASIC and of the operating system went hand-in-hand. Extant notes, memoranda, and recollections are concerned more with the design and development of the operating system than with BASIC itself. On the other hand, the project had a single set of goals, and these goals applied as much to the design of BASIC as to the design of the time-sharing system. For example, Memo No. 1 (Kurtz, 1963c, Nov.) reads:

> The Time Sharing System should be externally simple and easy to use for the casual programmer . . . .

A little later, the same document states:

> . . . flexible, including compiling in standard languages. However, no attempt will be made at making the use of the Time Sharing equipment be compatible with standard use of a computer.

(Presumably, "standard" meant "batch.") This latter quote sets the record straight: the Dartmouth system was always a multilingual system, even in its conception. And finally, on p. 2 of that memo:

> In all cases where there is a choice between simplicity and efficiency, simplicity is chosen.

Need I say more.

## 2.2. BASIC and Time Sharing

In addition to reflecting a common set of goals, the language BASIC and the time-sharing system in which it resides were, and still are, inextricably meshed. It is impossible for us to talk about one without the other. For example, each statement in BASIC begins with a line number, primarily to aid editing. To cooperate, the operating system monitor assumed that all inputted lines of text that started with a digit were part of a program. In addition, all programs were *automatically* sorted by line number prior to their being listed or run. Additions, deletions, and changes to programs could thus be made without overtly involving an editor. On the other hand, all inputted lines of text that started with a nondigit were assumed to be monitor commands. The discrimination between program lines and commands occurred at the earliest possible juncture in the operating system, rather than deeply within some editor, thus encouraging efficiency and rapid response. (A user wishing to create or change a text file lacking line numbers needed to overtly invoke an editor.)

A second example is the choice of monitor commands, such as HELLO, NEW, OLD, SAVE, LIST, RUN, and GOODBYE. Short common English words as system commands encouraged computer use by nonexperts. The true impact of this choice is almost beyond comprehension to a computer specialist, who is quite at home with "logon," "catalog," "execute," and "logoff," or their single letter abbreviations that are so common. Persons who use BASIC but don't understand the distinction between a language processor and an operating system monitor are incredulous when they learn that NEW, OLD, and RUN are not part and parcel of BASIC. The moral is obvious: simple and understandable command languages, like simple programming languages, are essential if nonexperts are to use the computer.

A final example is that the user deals directly only with his BASIC program. He need not even know that such things as "object code" exist. The user could compile (by typing RUN), receive error messages, edit by typing line-numbered lines, and recompile, all within seconds. Execution would automatically occur if there were no errors. This approach obviously required that compilation be fast, which in turn meant that it had to be single-pass. BASIC was always, and still is, a language simple enough to permit single pass compilation. Forward GO TO references and functions defined near the end of a program could be handled with linked lists and a "clean up" pass to fill in the missing transfers.

We make the point later that BASIC originally was not a true interactive language, since the INPUT statement that allowed data to be entered from the terminal during execution was not added until later. But BASIC lives, and has always lived, in interactive time sharing environments. While conceptually possible, a "batch" version of BASIC that used punched cards would be unthinkable.

## 2.3. Other Influences on BASIC

The relation of BASIC to FORTRAN and ALGOL is apparent. From FORTRAN comes the order of the three loop-controlling values—initial, final, step—thus permitting the step size to be omitted if it is unity. From ALGOL come the words FOR and STEP, and the more natural testing for loop completion *before* executing the body of the loop. These and other similarities are not surprising, since we knew both languages, and we did not hesitate to borrow ideas.

We also knew about the interactive language JOSS, but we preferred to stay nearer the main stream of "standard" languages. Several JOSS conventions—decimal points in line numbers, and periods after statements—did not appeal to us. Kemeny and I also had conversations with Richard Conway and others from Cornell about CORC, their "friendly" student-oriented language designed for card input (Conway and Maxwell, 1963). Like BASIC, CORC required the word LET for an assignment statement and made no syntactic distinction between integers and "real" numbers. Unlike BASIC, CORC simplified data input by fixing the format, provided an approximate equality, included a simple block structure, and required declaration of all variables. Interestingly, the educational goals stated in the article are similar to Dartmouth's, except that CORC was intended primarily for engineering students while BASIC was directed to liberal arts students.

BASIC was also influenced by the earlier DARSIMCO, which was really an approach to assembly language programming. For example, a BASIC program consists of a number of "instructions." Each instruction consists of three parts: the "instruction number," the "operation," and the "operand." This similarity is exhibited in the first BASIC program to appear in published form (Kemeny and Kurtz, 1964a, p. 4):

```
Instr. no.  Operation    Operand
   10       LET          X = (7+8) / 3
   20       PRINT        X
   30       END
```

## 2.4. Compiling versus Interpreting

While its implementation is not strictly part of a language, it is important to realize that Dartmouth BASIC was *always* intended to be compiled. A strong argument favoring an interpretive environment concerns "direct" versus "indirect" mode; that is, a user can enter a statement for direct execution, or change the program, *after* execution of the program has begun. This feature was never considered important for BASIC. We were concerned lest interpretative execution times would be prohibitive for any but the smallest programs. Instead, BASIC was intended to be a simple compiler-based language designed for use with terminals rather than cards, and for quick turnaround rather than batch-type delays.

The bias toward interpretive language processing in interactive environments was probably also influenced by the slowness of many compilers, which employed multiple passes. (One horror story claims that a certain FORTRAN compiler required 45 seconds to compile a program consisting *only* of the END statement; this was because the termination routine was located at the end of a long subroutine tape.) We estimated that a BASIC "compile" would require no more time than a "linking load". Load-and-go compiling placed a limit on program size, since the entire object code was assumed to be present in memory; this limitation proved more theoretical than real. As BASIC grew, program sizes were allowed to grow without conceding the single-pass ideal.

Another argument advanced in favor of interpreting involved instant syntactical error analysis. This approach went too far, in our view, and posed too great a load on the computer. In addition, it required users to compose programs in a restricted manner, giving them error messages before they had a chance to make obvious corrections on their own. BASIC at Dartmouth *never* checked for errors line-by-line. Instead, our strategy sepa-

rated program construction from compiling. If there were errors, reasonable error messages in English, limited to five in number per compilation, were produced only after the user requested a run. For small programs, this response occurred within five or ten seconds.

Despite the fact that both the Dartmouth and the General Electric implementations use compilers, BASIC is widely believed to be an interpretive language. Perhaps the reason is that most minicomputer implementations of BASIC use interpreters, for good reasons only loosely related to BASIC itself. Furthermore, minicomputers became quite widely available, and were the principal agents in the spread of BASIC. For instance, all of these minicomputers used interpreters for BASIC: the HP 2000, the DEC PDP/8, the DEC PDP/11, the DG NOVA, the WANG, and even the early DEC PDP/10. But BASIC itself, particularly at Dartmouth, possesses few properties that would a priori incline it toward being interpretive. In fact, its closest relative in many ways is FORTRAN, which was never thought to be interpretive.

## 3. Descriptions of BASIC

This section details the features of the sequence of BASICs developed at Dartmouth. The early versions were the most influential, but later versions are included to illustrate how the language has grown without sacrificing the original goals. What we omit, with deep regret, is the study of the many versions of BASIC developed elsewhere. Features have been catalogued for the more commonly available versions (Isaacs, 1973), and the diversity seems greater than the unity. All of these versions should however receive credit for helping to make computing more accessible to more people, which is the unifying theme that underlies BASIC.

### 3.1. BASIC, the First Edition

The first version of BASIC (Kemeny and Kurtz, 1974a) is similar to the proposed American National standard Minimal BASIC (ANSI, 1978). It included the statements or "instructions": LET, PRINT, END, FOR, NEXT, GOTO, IF THEN, DEF, READ, DATA, DIM, GOSUB, RETURN, and REM. A variable name consisted of a single letter or a single letter followed by single digit. (This restriction did not seem severe to us for two reasons: first, composing programs is quicker if shorter names are used; second, being mathematicians, we were accustomed to short variable names. As I recall, two arguments for short variable names were: we were concerned about symbol table lookup time, and we wanted space independence so that, for instance, `FOR I = 1 TO NSTEP2` would be recognized as `FOR I = 1 TO N STEP 2`.) Subscripted variables were named by single letters and could have one or two dimensions.

Expressions, which were called "formulas," were governed by the usual rules. The exponentiation operator was the up-arrow ↑, a symbol then available on a Teletype. This symbol was selected " . . . since on a teletype typewriter it is impossible to print superscripts, . . ." (Kemeny and Kurtz, 1964a, p. 5). The exponentiation operation A ↑ B took the absolute value of A, no doubt in order to calculate it by `EXP(B*LOG(ABS(A)))`. The manual explained that to compute X ↑ 3 when X could be negative, one should use X*X*X. Through an error in the treatment of unary minus signs (Kemeny, personal com-

munication, 1978 Feb. 7), $-X \uparrow 2$ was incorrectly interpreted as $(-X) \uparrow 2$. This was corrected in the third edition of BASIC.

Ten intrinsic functions were provided: SIN, COS, TAN, ATN, EXP, LOG, SQR, ABS, RND, and INT. The first four had the usual definitions and used radian measure. The next two used base *e*, not base 10. SQR provided the square root of the absolute value. ABS was of course the absolute value function. RND provided a "random" number between 0 and 1, but was quaint in one way: no doubt we were merely being lazy when we required that RND have a dummy argument. Although in later versions we quickly removed this unfortunate syntax, the damage had been done, and our bad choice later resurfaced in several ugly forms. The first version of INT truncated toward zero. That is, ". . . `INT(3.45) = 3, INT(-34.567) = -34`, . . " (Kemeny and Kurtz, 1964a, p. 31). INT was changed in the third edition of BASIC to conform to the "floor" function, in order to allow rounding by using INT(X + .5) for both positive and negative values of X. The GE-225 word size (20 bits) seduced us into making a third unfortunate decision—all function names had to have three letters, 20 bits being enough for three six-bit characters.

Regarding variable types, we felt that a distinction between "fixed" and "floating" was less justified in 1964 than earlier. The ratio of the execution times between floating and fixed calculations was diminishing. Furthermore, to our potential audience the distinction between an integer number and a non-integer number would seem esoteric. A number is a number is a number. Since all numeric variables were of type "number," there was no need either for explicit typing, as in ALGOL, or implicit typing, as with the FORTRAN IJKLMN convention.

We now consider the individual "instructions." The assignment statement required (and still requires) the word LET. We felt the assignment idea to be particularly tricky, and the word LET helped. `LET X = X + 1` is easier for novices to understand than is `X = X + 1`. Furthermore, we felt it important that *all* BASIC statements begin with an English word. The "instruction" END was required to be the last statement in a program.

The PRINT statement allowed expressions, used the comma as a separator, and allowed quoted material. The comma was "a signal to move to the next print zone (or to a new line if we are in the fifth zone)" (Kemeny and Kurtz, 1964a, p. 25). The comma could be omitted when the quoted material was combined with a simple variable or expression, as in `PRINT "FIRST NO. =" A, "SECOND NO. =" B` (Kemeny and Kurtz, 1964a, p. 25); later versions of BASIC required either a comma or a semicolon between the quoted material and the expression.

A cornerstone of BASIC is that numbers are automatically printed in a fixed format, thus freeing the user from having to learn about formatting. The format used depended on the number. If the number happened to be an integer, it was printed that way, without a decimal point. If the number was not an integer, but could be represented in a fixed decimal format with the decimal point adjacent to one of the six significant figures, it was so printed. Otherwise, an exponential format was used, with six significant digits and a power of ten. Six significant digits, which was within the accuracy of our hardware, was felt to be adequate for most applications.

The FOR and NEXT statements were designed to be more general than the FORTRAN DO and CONTINUE, but without allowing recalculation of the stepsize or final value, as permitted in ALGOL. The running variable is of course available to the programmer. We allowed noninteger values in the loop statement. This decision caused us fewer serious

problems than it might have, for the reason that arithmetic was done in binary floating point, and the results truncated rather than rounded. Thus, for instance, the value 0.1 was internally slightly less; consequently, a typical FOR statement `FOR X = 1 TO 10 STEP .1` worked as the unsuspecting programmer expected; namely, it provided 101 values of X = 1, 1.1, . . . , 9.9, 10. But since negative numbers on the GE-225 were represented as two's complements, −0.1 truncated became slightly larger in absolute value, and the construction `FOR X = 10 TO 1 STEP −.1`, fortunately less common, would *not* provide values of X = 10, 9.9, . . . , 1.1, 1.

The GOTO statement requires some explanation. We had already decided that BASIC programs would have line numbers. Why not, then, have the GOTO statement use the already existing line numbers? For those who program mentally, forward-pointing GOTOs required picking line numbers for later portions of the program. But this nuisance seemed preferable to defining a new syntactical entity called "label", and explaining this to students. Whatever the strengths or weaknesses of this decision, it was made, it is simple, and it *is* BASIC.

The IF-THEN statement allowed a relational expression using any one of the six relational operators: =, <, >, >=, <=, <>. The relational expression did not have to be enclosed in parentheses, as it did in FORTRAN. The two-way branching was pedagogically simpler than the three-way IF statement in FORTRAN.

The first version of BASIC allowed two methods for aiding program organization: the DEF defined function and the subroutine. DEF could be used to define up to 26 additional functions, with three-letter names ranging from FNA to FNZ. These had to have exactly one argument, and were restricted to a single line. They would appear anywhere in the program.

Subroutines were provided by a subroutine jump, GOSUB, which used a stack for the return address, and a RETURN statement to effect the return. In violation of present theories about program structure, a subroutine could have multiple entry points, and multiple exit points, but could not pass arguments. Some of us have learned how to use GOSUB with discipline, but BASIC itself remains laissez faire on this issue.

READ and DATA operated in an obvious manner. The interesting point is that READ was the *only* input statement in the first version of BASIC. Thus, not only was the first version of BASIC not interpretive, it wasn't even interactive!

The REM statement was included. We of course knew that FORTRAN had its C statements and ALGOL its comments, but the reason is not recorded for choosing the statement name REM or REMARK in BASIC.

The DIM statement for specifying array bounds was optional. In the absence of a DIM statement for an array, BASIC provided default subscript ranges of 1–10 in each dimension. The terms *vector* and *matrix* were used here. In later editions of the manual, we used the less mathematical *list* and *table*.

The conventions for creating and editing programs using line numbers were defined in this manual. One could SAVE programs, RUN (compile and go), and LIST them, but not much more. It was suggested that debugging could be aided by inserting extra PRINT statements. To start a new program, one had to type HELLO and reenter the user number, which was the student's regular ID number. Running time was computed to the nearest second, so that the trailing message "TIME: 0 SECS." was common, and the manual warned the user to "not be shocked" by it (Kemeny and Kurtz, 1964a, p. 17). The user was admonished that "since the space on the disks is limited, users are asked not to

save a program unless they really expect to use it again" (Kemeny and Kurtz, 1964a, p. 18).

The mathematical flavor of what was done is reflected in the examples appearing in the first manual: average of 100 logarithms, quadratic equation solver, maximum value of a function, Simpson's Rule, product of two matrices, binomial probabilities. The program that found the maximum value of a function took "1 MINS. 11 SECS." on the GE-225 (Kemeny and Kurtz, 1964a, p. 37). (A penciled notation states that the same problem in FORTRAN on a 1620 model II required 4 minutes 55 seconds.) In the manual we promised that with the forthcoming GE-235, the problem should take less than 10 seconds. Just for fun, that same problem required 1.181 seconds in 1977 (in Dartmouth BASIC on a HIS 66/40 computer). Finally, the manual promises that "a second language, MATRIX, will be designed specially for matrix work" (Kemeny and Kurtz, 1964a, p. 39).

### 3.2. BASIC, the Second Edition

Starting with the third edition of BASIC, each new edition of the manual signaled a significantly different version of BASIC. Since 1966 we have therefore identified the versions of BASIC by the edition number of the manual. On the other hand, the first and second editions of the manual each referred to the original version of BASIC, with only minor changes. The manuals themselves differ in that the first edition was organized as a reference manual while the second edition was organized as a primer, reflecting BASIC's principal clientele.

One other historical fact: although neither the first edition nor the second edition was actually signed, my memory records that John Kemeny wrote the first edition while I wrote the second edition.

The manual for the second edition (Kemeny and Kurtz, 1964b, Oct.) bears a date only five months after that of the birth of BASIC. In that brief time, the semicolon was added as an allowable print separator. Its purpose was to "pack" numeric output by using only the actual spaces needed, up to the nearest multiple of three spaces. (This unfortunate glitch, rectified in later versions, can be traced in non-Dartmouth implementations.)

The other important addition was that the zero subscript, as in A(0) or B(0,0), was now allowed. Polynomial coefficients could now be represented naturally as, for instance, C(0), . . . , C(N).

A slightly different version of BASIC called CARDBASIC was also described in the second edition. It was designed for card input and printer output, and operated in background, which shared time slices with the foreground users. CARDBASIC contained the first version of the MAT statements, which had been promised in the first edition. These statements appeared in regular BASIC in the third edition, and are therefore described in the next section. There were of course other features, such as allowing longer programs and more data. CARDBASIC differed in one other important way: it did not allow the zero subscript. CARDBASIC lived a short life, since one of our goals was to eliminate dependence on punched cards.

I make one last observation about the second edition. Long before the current surge of interest in structure and style in programming, the following sentences appeared (p. 48):

> There are some matters that do not affect the correct running of programs, but pertain to style and neatness. For instance, as between two or more similar ways to prepare a part of a program,

one should select the one that is most easily understood unless there is an important reason not to do so.

And, after some hints about data organization, the following appeared (p. 49):

> No doubt the user will be able to devise other ways to make a program neat and readable. But again, the important consideration in style is to program in a way that makes it more understandable and useful to both oneself and others in the future.

These sentences give little hint as to what the user was supposed to do in order to achieve good style, but at least the thought was there. Since then several stylistic ideas have been tried, and there is now a book about structure and style using elementary BASIC (Nevison, 1978).

### 3.3. BASIC, the Third Edition

There were only two major changes to Dartmouth BASIC in the 15 months since the second edition. By 1966 we knew our efforts would shortly be directed to a GE-635 computer, and we therefore felt that further development of BASIC for the GE-265 was pointless. The manual (Kemeny and Kurtz, 1966) records a few minor changes. A SGN (signum) function was added, and it was noted that repeated exponentiation was left-associative. A RESTORE statement was added to reset the internal pointer to the data block, which was constructed from the totality of the DATA statements. The precedence of the unary minus sign was finally corrected (p. 40) so that $-X \uparrow 2$ would be interpreted as $-(X \uparrow 2)$.

The first major addition was that of the INPUT statement, which accepted data from the terminal during execution. BASIC was at last an interactive language!

The second major addition was the collection of MAT statements, originally tested in CARDBASIC (see the previous section). These allowed operations on arrays of one and two dimensions. When it mattered, a vector—an array of one dimension—was treated as a column vector. The following operations were described:

```
MAT READ A, B, C
MAT PRINT A, B; C
MAT C = A + B
MAT C = A - B
MAT C = A * B
MAT C = INV(A)
MAT C = TRN(A)
MAT C + (scalar expression)*A
MAT C = ZER
MAT C = CON
MAT C = IDN
```

In MAT PRINT, the semicolon caused packed printing while the comma caused printing in zones. The * stood for matrix multiplication, not element-by-element multiplication. INV and TRN produced an inverse and transpose, respectively. ZER and CON produced arrays of zeros and ones, respectively, while IDN produced an identity matrix. The statement `MAT READ A(M,N)` caused the array A to be redimensioned before the reading took

place. Redimensioning was also allowed with the ZER, CON, and IDN functions. `MAT A + B*A` and `MAT A = B` were specifically disallowed. It was stated (p. 47) that the former would "result in *nonsense*" and that the latter could be achieved by the statement `MAT A = (1)*B`.

The MAT statements in CARDBASIC (Kemeny and Kurtz, 1964b, Oct.) had dealt with arrays whose subscripts started with 1. In the third edition (Kemeny and Kurtz, 1966), subscripts started with 0, and the MAT statements were changed accordingly. Thus, for example, CON(3,4) was a four-by-five array of ones. This also meant that the upper left hand corner of a matrix was the element subscripted (0,0). Needless to say, we were not completely happy with our choices.

On a happier note, here was the first appearance in BASIC of indentation to suggest the scope of a loop (p. 19).

### 3.4. BASIC, the Fourth Edition

By 1965–1966, system use had grown to about thirty simultaneous Teletype model 33 and 35 terminals, a number of which were off campus in secondary schools and colleges. Plans were afoot to replace the hardware with a GE-635 and move to a new building. Although the third edition continued in use into the fall of 1967, a new experimental version of BASIC appeared on the GE-635 in the spring of 1967. It was described in the Supplement to the BASIC Manual, Third Edition (Kemeny and Kurtz, 1967).

The software for the new hardware was developed jointly by General Electric and Dartmouth; GE wrote the operating system while Dartmouth wrote the compilers and editors. Both General Electric and Dartmouth contributed to the design of this version of BASIC. For instance, GE made suggestions concerning strings and files, presumably to improve compatibility with their Mark I BASIC (phone conversation, Neal Weidenhofer, 1978 February 15).

The Supplement notes that the previously required dummy argument for the RND function was omitted, and a new statement, RANDOMIZE, was added to "randomize" the starting point of the random number sequence. Defined functions were now allowed to have no arguments. The ON-GOTO statement was added as a multiple-way branch. In order to "increase the similarity between the ON and IF-THEN" all variations—IF-THEN, IF-GOTO, ON-THEN, and ON-GOTO—were permitted. The argument of the ON-GOTO was truncated and, if out of range, caused a program halt.

The semicolon PRINT separator now produced contiguous printing. A TAB function was added, with the column positions numbered from 0 to 74. Multiple assignments in a single LET statement were allowed, as were assignments to subscripted variables having subscripted variables as subscripts. For the first time, BASIC initialized all variables to zero.

Two major additions appeared. The first provided string variables and arrays of strings (one-dimensional only) together with all relational operators, thus allowing the sorting of alphabetical information. The DATA statement was expanded to allow unquoted string constants as data. String data and numeric data were retained in two separate pools, regardless of how they might be intermixed in the source program. A statement RESTORE$ was provided to reset the pointer to the string data. (RESTORE* reset the numeric data pointer, while RESTORE reset both; this feature persisted until the sixth edition of

BASIC in 1971.) One serious error was made, with the good intention of making things easier for the casual programmer—trailing spaces were ignored in string comparisons. Thus "YES" and "YES ␣" were treated as equal. (This feature persists into Dartmouth's current BASIC, but will almost certainly disappear in the next version.)

The other major change was in the MAT package. There were a few minor additions, such as allowing `MAT A = B`, and various options for MAT PRINT. But the important change was that MAT now ignored the zeroth row and column. A programmer could now expect to find the "corner" element of a matrix in position (1,1). This compromise has worked well over the years, despite theoretical and esthetic arguments against it. One statement, `MAT INPUT V`, where V is one-dimensional array variable, was added to permit arbitrary amounts of input, separated by commas. Though not strictly speaking a MAT operation, this capability was urgently needed. A zero-argument function NUM was added to provide the actual number of data entered.

The manual for the fourth edition finally appeared (Dartmouth, 1968). Since most of the new features had been described in the Supplement to the Third Edition (Kemeny and Kurtz, 1967), only a few additional features now made their appearance.

Multiple-line defined functions were allowed, though they still accepted only numeric arguments and produced only numeric values. The CHANGE statement was added to change a string to a numeric vector consisting of the ASCII codes for each character of the string, and vice versa. The zeroth element of the vector contained the number of characters. This capability proved extraordinarily useful, since any conceivable string operation could be carried out by equivalent operations on numbers.

### 3.5. BASIC, the Fifth Edition

This, the first version of BASIC to exist with the DTSS operating system, was described initially in two supplements to the fourth edition of BASIC (Dartmouth 1969 February, April). The fifth edition manual appeared in late 1970 (Dartmouth, 1970).

Important changes appeared in this version of BASIC. The most significant was the addition of files. DTSS was designed around a random access file system. The catalog entry for each file included a number of "device addresses" so that a file could be distributed over several devices. The address of any particular point in the file could be calculated easily from these device addresses. We were thus able to allow both sequential files, which we called "terminal format" files, and random access files. The latter could contain either numeric or string data, but not both. A FILES statement associated filenames with file reference numbers, which were really the ordinal numbers of the filenames in the FILES statement. (This stringent early binding requirement was relaxed with the FILE statement, described shortly, and was eliminated in the sixth edition of BASIC.) For random access files, the file type was indicated explicitly in the file name. Numeric file names required a "%" following the filename proper, while string files require a "$xx", where xx was a constant integer that gave the maximum string length.

The FILE statement was introduced to allow changing the file name associated with a particular file reference number. This more natural file opening mechanism became more commonly used, even though the FILES statement remained; the user merely specified unnamed scratch files in the FILES statement and specified his real files in several FILE statements.

In order to help casual programmers distinguish between sequential and random-access files, separate statements were provided. The PRINT and INPUT statements applied to sequential files. IF END was used to determine the end-of-file condition. Both input and printing operations could take place on the same file.

For random-access files, the principle statements were READ and WRITE. The location of the file pointer was given by a LOC function, and the file length by LOF. The RESET statement for this type of file could set the pointer to any location in the file. The end of the file was detected by a statement such as:

```
IF LOC(#1) >= LOF(#1) THEN <process end of file>.
```

Two program segmentation features were added in this version. The simplest was CHAIN, which allowed a program to initiate the execution of another program. CHAIN permitted almost any system structure, although it was inefficient at times. It could, for instance, allow a program to create a whole new program, and then transfer to it. At this stage, only simple chaining was allowed, though later versions allowed file passing.

The other segmentation feature was called subprograms, although it was really a variant of overlaying. The variable names of the several segments were common, as if in a giant COMMON, but the line numbers were independent. These so-called subprograms were referred to by reference number, and obviously had to be compiled with the main program. There were a number of petty restrictions with this feature, and by hindsight it is easy to conclude that this was a bad idea. It did serve, however, to allow construction of moderately large systems using BASIC. The most notable was the first version of IMPRESS, a system for analyzing social science survey data (Dartmouth, 1972).

These two segmentation features were similar to features designed by GE for its Mark I BASIC on the GE-265 computer. It is not known whether Dartmouth devised them independently or copied them from GE.

This version contained corrections of previous minor flaws, such as now leaving the for variable untouched if the for loop was to be executed zero times. Two-dimensional string arrays were added. The individual characters of a string could be obtained only through the CHANGE statement. Defined functions could now have string arguments and values. Several new functions were added: CLK$, DAT$, LEN, USR$, STR$, VAL, ASC, and TIM. They provided, respectively, the wall clock time, the date, the length of a string, the current user number, the printed form of a number but without leading or trailing spaces, the numeric value of a string presumed to represent a number, the ASCII number of a character, and the elapsed running time of a job. In-line comments appeared for the first time, using the apostrophe as a separator.

The CHANGE statement was amplified to allow specifying the bit size of the characters. Thus, `CHANGE A TO A$ BIT 4` allowed the string A$ to contain decimal digits, each requiring only four bits, thereby obtaining greater packing densities for data. (This feature was added to accommodate Project IMPRESS.)

User defined functions were allowed to be recursive, but were implemented so inefficiently that they were (arbitrarily) prohibited from being recursive in the subsequent version of BASIC.

In the MAT package, `MAT A = A*B` was still not allowed.

### 3.6. BASIC, the Sixth Edition

In the fall of 1969, even prior to the appearance of the manual for the fifth edition of BASIC, we began designing the sixth edition. In addition to Kemeny and myself, the group included three former students who had returned to Dartmouth as faculty: Stephen Garland, John McGeachie, and Robert Hargraves. It is probably the best-designed and most stable software system Dartmouth has ever written. Previous versions had been hastily designed, introducing many flaws to be corrected in subsequent versions, which appeared often. This time we designed thoughtfully and slowly. Ideas were carefully discussed. Specifications were written down in advance. Undergraduate programmers Kenneth Longmuir and Stephen Reiss began their work only after the design was essentially complete. The compiler and runtime support routines were extensively tested, and were operated as an experimental version for over three months during the summer of 1971. The manual was prepared in advance and published *prior* to the official installation of the new version (Mather and Waite, 1971). A formal specification was prepared; it appeared two years later (Garland, 1973). This new version went into regular operation on September 21, 1971. There were almost no problems, conversions having taken place during the summer. This stands in our mind as a rare and shining example of how a major system change should take place.

Though not part of the ancient history of BASIC, the design of the sixth edition at Dartmouth is discussed here for several reasons. First, it really is the culmination of the original BASIC project, having grown from the rapidly changing earlier versions. Second, it appeared first in 1971, which was seven years from the initial version, and which was about seven years ago. Third, it has been used since 1971 without significant change, a rarity among university-built software.

Major changes were made to subprograms, files, the string package, and output format control. (Minor changes were also made, but these will not be discussed here.)

The most significant change was the scrapping of the previous subprogramming feature, and its replacement by subprograms that connect to the calling program only through the calling sequence. The statements used are CALL, SUB, and SUBEND. Parameters are specified by reference. There is no syntactical distinction between input and output parameters. Unlike early FORTRAN, parameters and arguments are matched at each CALL for correct type and number. Unlike ALGOL, parameters cannot be specified "by name," nor can there be global variables. Since the subprograms exist independently of the calling program, they can be separately compiled and collected into libraries. They can also overlay one another, thus allowing BASIC to be used for constructing large systems. Indeed, the management information system (FIND) used at Dartmouth is written almost entirely in BASIC (McGeachie and Kreider, 1974).

The second major change was to files. The FILES statement was dropped completely. The FILE statement provides execution-time binding of file names to file reference numbers. The requirement for file typing by trailing character was removed, since BASIC can almost always tell what the user wants. The ambiguous case occurs in writing to an empty random-access string file; here, the MARGIN statement specifies the maximum string length for such a file. Two new functions were added to permit file type checking by the programmer. In the previous version, a nonexistent file name appearing in a FILE statement would cause termination. In this version, no error can occur on a FILE statement. A

program can then use special functions to determine if the file actually exists, and what operations are permitted on it. Naturally, if an inconsistent operation is attempted, termination still occurs.

That we should expand string manipulation in BASIC was clear. Less clear was what substring notation should be used. We therefore chose to use string functions, since they could be changed more easily than new syntax. The SEG$ function provides a substring, while the "&" causes string concatenation. The POS function can locate a given string expression in a string. Together, these functions allow all string operations. For example, assignment into the interior of a string can be done by constructing the desired string with SEG$ and &.

A form of "image" formatting was added. It is employed by including in the PRINT statement a string expression which specifies the image (rather than the statement number of an "image" statement).

A final note: `MAT A = A*B` is now legal.

### 4. Dartmouth and General Electric

The history of Dartmouth time sharing and the BASIC language is intertwined with the early history of the General Electric time-sharing business. It is therefore appropriate here to outline key events in the relationship. This will not be a complete history, nor will it extend credit to all that are due it.

Our first technical contact with General Electric occurred in the summer of 1962. Anthony Knapp (a student) and I visited GE in Phoenix and examined the GE-225, the Datanet-930 (a predecessor to the Datanet-30), and the dual access disk controller. On returning to Dartmouth, Knapp sketched, in two weeks, an outline of a time-sharing system using that equipment. A descriptive document entitled MESS (Knapp and Kurtz, 1962) was mailed to everyone we had met, no doubt in the hope that GE would immediately want to give us a computer on which to carry out our ideas. MESS elicited no response.

Our own plans nevertheless gathered steam, and in April of 1963 Dartmouth sent to General Electric a letter of intent to purchase a system. During the summer of 1963, John Kemeny developed a compiler for a draft version of BASIC, using GE-225 computers in the New England area. Our own hardware arrived in February of 1964, with assists from GE educational discounts and two NSF grants made in January 1964.

During the spring and summer, persons from the Engineering Department at General Electric in Phoenix followed our progress. They provided both moral and material assistance—a FORTRAN compiler and additional magnetic tape drives, for example. In the fall, Dartmouth demonstrated at the Fall Joint Computer Conference in San Francisco, with General Electric providing the costs of the display. In December of 1964 and January of 1965 a copy of the Dartmouth system, including BASIC and ALGOL, was installed in the Engineering Department in Phoenix.

Shortly thereafter, operation of that machine was transferred to GE's already-established service bureau business. Since the hardware was a marriage of the GE-235 and the Datanet-30, the new system was named the GE-265. Additional copies were put into operation as their service bureau business grew.

In 1965 we attempted to obtain support from GE for our continuing software work. Louis Rader, then a Vice President of General Electric, responded in September of 1965 by offering to place an entire GE-635 system in the Kiewit Computation Center then under

construction. The terms included joint use of the machine for three years, and a joint project to develop an operating system for that machine. GE was to retain title to the machine for the period of the agreement, which started on October 1, 1966.

While Dartmouth ceased in 1965 enhancing its third edition version of BASIC (see Section 3.3), GE added important features to its version for the Mark I service. These features included strings and files (phone conversation, Derek Hedges, 1978 February). They also designed a CHAIN statement, borrowed from an early version of FORTRAN, and a CALL statement, which was similar to the CALL feature found later in the fifth edition of Dartmouth BASIC (see Section 3.5).

Software teams from both sides formed in the spring of 1966. The Dartmouth team included Kemeny, Kenneth Lochner, and myself, together with students Sidney Marshall, David Magill, Sarr Blumsen, Ronald Martin, Steven Hobbs, and Richard Lacey. A GE-635 computer at RADC, Griffiss AFB, Rome, New York, was used in the summer and fall to develop and operate MOLDS, a multiple user on-line debugging system designed by Marshall.

A GE-635 was installed at Dartmouth in late 1966 and early 1967. Using MOLDS, GE personnel developed a Phase I operating system for this machine. Dartmouth developed the fourth edition of BASIC to specifications prepared jointly with GE, and provided communications software. This work continued through 1967 until Phase I became operational in September of 1967, allowing Dartmouth to sell its GE-265. Both Dartmouth people and GE customers shared the new machine under the terms of the joint agreement.

In April of 1967, having completed its part of the Phase I development, Dartmouth began designing its own "ideal" operating system, which was called Phase II. Joining the group at this time were former students Stephen Garland and Robert Hargraves. Marshall continued but with the new title Chief Designer. The project group, which included many other persons, met weekly to resolve problems; Kemeny served as chairman and final arbiter.

General Electric established additional sites in the fall of 1968, and transferred its commercial customers to them. Dartmouth continued to use Phase I until late March 1969, when Phase II replaced it. The name Phase II was changed to DTSS shortly thereafter.

The fourth edition of Dartmouth BASIC (see Section 3.4) was the original version on the GE Phase I operating system, which shortly became the Mark II service. GE continued to modify and use that version of BASIC, whereas Dartmouth replaced it in 1969 with its own fifth edition of BASIC.

After 1967 there was little exchange of software with GE, though both sides made numerous good faith efforts to exploit the association and achieve mutual benefits. The title to the GE-635 passed to Dartmouth in September of 1969. An additional three year agreement was then executed, but little occurred under its terms other than the gift from General Electric to Dartmouth of several additional major pieces of GE-635 hardware. All contractual association with GE ceased on September 30, 1972.

While working together from 1966 to 1968 posed problems for both sides, there is no question that each benefited enormously from the association. Dartmouth obtained a large machine far beyond its own means at that time. General Electric started in time sharing initially by adopting the original Dartmouth system (Mark I on the GE-265), and later shifted to a larger system (Mark II on the GE-635) through close association with Dartmouth.

## 5. Evaluation

### 5.1. A Priori Evaluation

Our goal was to provide our user community with friendly access to the computer. The design of BASIC was merely a tool to achieve this goal. We therefore felt completely free to redesign and modify BASIC as various features were found lacking. We have always remained loyal to our overall goals, while at the same time we allowed the language to grow to meet the increasingly sophisticated tastes. We did not design a language and then attempt to mold the user community to its use.

Little a priori evaluation of BASIC exists. A memorandum (Kemeny and Kurtz, 1963) prepared for internal use discussed time sharing, simple user interfaces, and open access computing, but failed to mention BASIC, and made only a passing reference to the need for a "simple language." Although two proposals were submitted to the NSF in 1963 and funded in 1964, neither dealt primarily with BASIC. One was a facilities proposal based on research, the other dealt with course content improvement. Understandably, there were doubts about our ability to develop a time sharing system using undergraduate students as programmers. Nonetheless, the two grants were made.

### 5.2. A Posteriori Evaluation

BASIC and the associated system commands spread rapidly. The original version, for the GE-265, was replicated in perhaps fifty sites, and for several years served General Electric as its Mark I time sharing system. Other users of the GE-265 included Call-A-Computer and several educational institutions.

The Dartmouth approach served as the model for two of the most widely used small time sharing systems ever developed: the HP 2000, and the DEC PDP/11-RSTS. Almost all small systems now offer BASIC, either exclusively or with other languages. Although one of COBOL or FORTRAN is probably the most widespread language in terms of number of lines of code ever written, it is my opinion that more people in the world know or have seen BASIC than any other computer language. I can only wildly guess that perhaps five million school children have learned BASIC on some minicomputer.

In 1973 when consideration was being given to the possible standardization of BASIC, we discovered that the number of commercial time-sharing service bureaus that offered BASIC was greater than the number offering any other language (SPARC, 1973). Some criticized that it was too late to standardize BASIC. Perhaps, but committee X3J2 has been one of the largest and most enthusiastic of the language standards committees. The standard for Minimal BASIC (ANSI, 1978) is in the final stages (as of 1978 February). Close cooperation with ECMA/TC21 (the European counterpart of X3J2) has ensured that the European and American standards are technically equivalent. Work continues on enhancements to Minimal BASIC.

Developing a standard for BASIC has suffered from the multiplicity of different versions. The major differences have been catalogued (Isaacs, 1973), and Isaacs admonishes the user to stick to elementary BASIC for transportability. Indeed, the standards committee X3J2 was charged to develop a standard for Minimal BASIC first, partly because it might fail if it tried to do more.

BASIC is truly a product of grass roots efforts. Hobbyists can and do write their own compilers, adding their own innovations. Rather than being a single language with dialects, BASIC is really a class of languages, all with a common core. Variations and features abound. From Dartmouth's point of view, some languages called BASIC really shouldn't be, since they violate one or more of our original criteria, although they may have borrowed other features. For instance, our dogma prohibits the optional LET, several statements on one line, and a single statement continued over several lines.

In some ways, the effort to standardize BASIC has suffered from the same malady that plagued the early FORTRAN standards efforts: the multiplicity of variations that had to be reconciled. If anything, the task is more difficult since BASIC grew unchecked for nine years (from 1964 to 1973) before the effort began. In contrast, APL was designed by a single person and copied almost exactly, so that it became its own standard. COBOL was designed as a standard language to begin with. ALGOL was also designed before it was used, so that the ALGOL 60 Report serves as its standard.

What would we change if we were to start over? We now know that spaces should be used to improve clarity, and to indicate scope through indentation. We also now know that the selection of variable names is far too limited. Requiring spaces around keywords, as ANS Minimal BASIC does, and multicharacter variable names go hand in hand. This change is high on our wish list.

We also now know the limitation of the primitive GOTO and IF THEN statements in BASIC. A version called SBASIC (Garland, 1976) dispenses with GOTO, IF THEN, GOSUB, and ON GOTO, all of which target to line numbers, and replaces them with the DO and LOOP for loops, the IF, THEN, ELSE, and CONTINUE for decision mechanisms, the PERFORM, DEFINE, and DEFEND for subroutines, and the SELECT CASE for multiple-way branches. While the particular constructs selected are not ideal in all cases, SBASIC has been taught in several elementary courses with very good results. My own opinion is that GOTO has an attractive simplicity for novices, but that it should be discarded at an early stage in favor of structured constructs.

Finally, to paraphrase, one graph is worth a thousand numbers. About 10% of all BASIC programs at Dartmouth produce graphical output. This being the case, graphical statements have been added to the language by means of a preprocessor. The statements include PLOTTER < device > , VIEWPORT, WINDOW, CLEAR, BORDER, AXIS, PLOT, and others. Also included are picture definitions similar to subprograms, simple ways of applying transformations such as translation and rotation, and three-dimensional perspectives (Garland, 1976; Arms, 1977; Hardy, 1978).

## 6. Implications

BASIC has become the most widely known computer language. Why? Simply because there are more people in the world than there are programmers. If ordinary persons are to use a computer, there must be simple computer languages for them. BASIC caters to this need by removing unnecessary technical distinctions (such as integer versus real), and by providing defaults (declarations, dimensioning, output formats) where the user probably doesn't care. BASIC also shows that simple line-oriented languages allow compact and fast compilers and interpreters, and that error messages can be made understandable.

## ACKNOWLEDGMENTS

I would like to thank the numerous persons who read early drafts of this paper, suggested changes, and filled lapses in my own faulty memory. John Kemeny especially recalled important facts and ideas. More than that, our close collaboration of 22 years has made it difficult to separate our respective contributions to the overall effort.

I would also like to thank Henry Ledgard and Ted Lewis, the reviewers, for their many helpful suggestions. My special thanks go to Stephen Garland, who kept better files than I concerning the early days, and to Kent Morton, who corrected numerous stylistic flaws. As is customary, the author accepts final responsibility for all remaining errors, factual and otherwise.

Finally, the author expresses his deep appreciation to Jean Sammet for organizing the History of Programming Languages Conference, thereby goading the author into writing about a subject that has appeared only rarely in the technical literature.

## REFERENCES

ANSI (1978). *American national standard programming language: minimal BASIC*, X3.60—1978. New York: American National Standard Institute.

Arms, W. Y. (1977). *Graphics with SBASIC* (unpublished memo). Hanover, New Hampshire: Kiewit Computation Center.

Conway, R. W., and Maxwell, W. L. (1963). CORC—the Cornell computing language. *CACM* **6**(6): 317–321.

Corbato, F. J., Merwin-Daggett, M., and Daley, R. C. (1962). An experimental time sharing system. In *AFIPS Conference Proceedings*, Vol. 21. Palo Alto, California: National Press.

Dartmouth (1956). *DARSIMCO*. Hanover, New Hampshire: Dartmouth College.

Dartmouth (1968). *BASIC*, 4th ed. Hanover, New Hampshire: Dartmouth Publ.

Dartmouth (1969a) February. *BASIC*, 4th ed. (supplement, version I). Hanover, New Hampshire: Kiewit Computation Center.

Dartmouth (1969b) April. *BASIC*, 4th ed. (supplement, version II). Hanover, New Hampshire: Kiewit Computation Center.

Dartmouth (1970) September. *BASIC*, 5th ed. Hanover, New Hampshire: Kiewit Computation Center.

Dartmouth (1972) July. *The IMPRESS manual*. Hanover, New Hampshire: Project IMPRESS.

Garland, S. J. (1973). *Dartmouth BASIC, a specification*. Hanover, New Hampshire: Kiewit Computation Center TM028.

Garland, S. J. (1976). *Structured programming, graphics, and SBASIC*. Hanover, New Hampshire: Kiewit Computation Center SP028.

Gries, D. (1968) January. Use of transition matrices in compiling. *CACM* **11**(1): 26–34.

Hardy, S. (1978) February. *Graphics in BASIC and structured BASIC*. Hanover, New Hampshire: Kiewit Computation Center TM110.

Hargraves, R. F. (1959) October. *DART I*. Hanover, New Hampshire: Dartmouth Computation Center.

Ingerman, P. Z. (1961) January. Thunks. *CACM* **4**(1): 55–58.

Irons, E. T. (1961) January. A syntax directed compiler for ALGOL 60. *CACM* **4**(1): 51–55.

Irons, E. T., and Feurzeig, W. (1961) January. Comments on the implementation of recursive procedures and blocks in ALGOL 60. *CACM* **4**(1): 65–69.

Isaacs, G. L. (1973). *Interdialect translatability of the BASIC programming language*. Iowa City, Iowa: American College Testing Program. ACT Tech. Bull. No. 11.

Kemeny, J. G. (1962) May. *Dartmouth oversimplified programming experiment*. Hanover, New Hampshire: Dartmouth Computation Center.

Kemeny, J. G., and Kurtz, T. E. (1963). *A proposal for a college computation center*. Hanover, New Hampshire: Dartmouth College.

Kemeny, J. G., and Kurtz, T. E. (1964a) June. *BASIC instruction manual*. Hanover, New Hampshire: Dartmouth College.

Kemeny, J. G., and Kurtz, T. E. (1964b) October. *BASIC*, 2nd ed. Hanover, New Hampshire: Dartmouth College Computation Center.

Kemeny, J. G., and Kurtz, T. E. (1966). *BASIC*, 3rd ed. Hanover, New Hampshire: Dartmouth College.

Kemeny, J. G., and Kurtz, T. E. (1967). *BASIC*, 3rd ed. (supplement). Hanover, New Hampshire: Dartmouth College.

Kemeny, J. G., and Kurtz, T. E. (1968). Dartmouth time sharing. *Science* **162:** 223–228.

Knapp, A. W., and Kurtz, T. E. (1962). *MESS, a master executive supervisory system.* Hanover, New Hampshire: Dartmouth College.
Kurtz, T. E. (1962a) February. *ALGOL for the LGP-30, a comparison.* Hanover, New Hampshire: Dartmouth Computation Center.
Kurtz, T. E. (1962b) March. *ALGOL-30 system operating instructions.* Hanover, New Hampshire: Dartmouth College Computation Center. Procedure Manual CCM-5.
Kurtz, T. E. (1962c) October. *A Manual for SCALP, a self-contained Algol processor.* Hanover, New Hampshire: Computation Center.
Kurtz, T. E. (1963a) January. *Programming for a Digital Computer.* Hanover, New Hampshire: Dartmouth College.
Kurtz, T. E. (1963b) May. *Languages* (unpublished memo). Hanover, New Hampshire: Dartmouth College.
Kurtz, T. E. (1963c) November. *Background for the time sharing system.* Hanover, New Hampshire: Dartmouth Computation Center Time Sharing Project Memo No. 1.
Loveday, E. (1977) September. George Stibitz and the Bell Labs Relay Computer. *Datamation* pp. 80–85.
Mather, D. G., and Waite, S. V. F., eds. (1971). *BASIC,* 6th ed. Hanover, New Hampshire: University Press of New England.
McCarthy, J., *et al.* (1963). A time sharing debugging system for a small computer. In *AFIPS Conference Proceedings,* Vol. 23, pp. 51–57. Washington, D.C.: Spartan Books.
McGeachie, J. S., and Kreider, D. L. (1974). Project FIND—an integrated information and modeling system for management. In *AFIPS Conference Proceedings,* Vol. 43, pp. 529–535. Montvale, New Jersey: AFIPS Press.
Naur, P., ed. (1960). Report on the algorithmic language ALGOL 60. *CACM* 3(5): 299–314.
Nevison, J. M. (1978). *The little book of BASIC style.* Reading, Massachusetts: Addison-Wesley.
Samelson, K., and Bauer, F. L. (1960) February. Sequential formula translation. *CACM* 3(2): 76–83.
SPARC (1973). *Proposal for an American National Standard project under committee X3, the BASIC programming language.* Washington, D.C.: CBEMA.
Wheelock, E. (1769). *The Dartmouth College Charter.* Portsmouth, New Hampshire: King George III.

## TRANSCRIPT OF PRESENTATION

THOMAS CHEATHAM: For the rest of the session, we'll turn our attention to the BASIC language. Professor Tom Kurtz is going to talk about that. In 1964 Tom was an Associate Professor of Mathematics, teaching mainly Statistics at Dartmouth College, and at the same time, the Director of the Dartmouth College Computing Center. Initially they had an LGP 30, [but] in the Spring of 1964 they got their first Dartmouth time-sharing system, and then on to BASIC. At the present point in time, Tom is a Professor of Mathematics, teaching about half-and-half in Statistics and Computer Science, having, I guess wisely, given up the Computing Center.

THOMAS E. KURTZ: This is the last talk of the day. I would like to believe that the plan was to save the best for last. I'm not sure about that, but I do have one firm principle myself and that is not to cut into the cocktail hour, so I'm in complete accord with the firmness of the management to hold the talks on schedule.

In a way, this is like a reunion for me, because I got my start in computing in [the summer of] 1951 at the National Bureau of Standards Institute for Numerical Analysis, which was located on the UCLA campus. As a matter of fact, there was one other person who was also a student in that program that summer; that was Ken Iverson. And I think it's kind of interesting that out of six summer students who were involved that year, and who happened to be students of a man named Forman Acton, two of them went on to invent or coinvent computer languages.