Charles Crume

The True BASIC Computer Language: An Introduction and Critique.

CHARLES CRUME is a Senior Technical Consultant, University of Nevada System Computing Services, Computing Center Building, Reno, NV 89557-0023.

The True Basic Computer Language: An Introduction and Critique

True BASIC is the latest implementation of the popular BASIC programming language (Davis, 1986; Kemeny & Kurtz, 1985c). BASIC was developed at Dartmouth College in the early 1960s by John G. Kemeny and Thomas E. Kurtz. These professors had observed beginning programming students for many years and decided that time sharing, rather than traditional batch programming techniques would be more appropriate for such students (Kemeny & Kurtz, 1985a). BASIC was therefore developed for use with computer time-sharing schemes. During their design of the original BASIC, the following criteria were developed (Kemeny & Kurtz, 1985a):

1. It should be easy to learn for the beginner.
2. It should be a general-purpose language, allowing the writing of any program.
3. Advanced features had to be added so that, if there was a price, it was paid by the expert, not the novice.
4. It should take full advantage of the fact that the user could interact with the computer.
5. It should give error messages that were clear and friendly to the user.
6. It should give fast response for small programs.
7. No understanding of the hardware should be necessary.
8. It should shield the user from the operating system.

Since the 60s, various dialects of the language have been developed. Although the original BASIC was (and still is) a compiled language, most dialects were implemented through interpreters (Kemeny & Kurtz, 1985a). (Interpreters are special programs that translate a single line of code into machine language, then execute that line before accepting the next line of code.) Interpreted versions were produced because most dialects were written for microcomputers that had limited amounts of main memory.

Many different dialects emerged for use on a variety of hardware. This happened for several reasons. Portions of the original BASIC language were frequently removed due to memory limitations of various computers. Then, too, machine dependent features were often added to take advantage of specific hardware configurations. Davis (1986) states: Unfortunately, no two BASICSs are quite the same. IBM BASIC differs from Apple BASIC, and the version that runs on a Radio Shack computer differs from

both of them. A program written in Apple BASIC won't run on an Atari computer, or on an IBM PC. Moving from one BASIC to another is difficult, as key features are implemented in subtly different ways. There is no single language called BASIC. Instead, we have a series of dialects, all derived from the original Dartmouth version. (p. iii) The resulting witches' brew of dialects have been dubbed Street BASIC (Kemeny & Kurtz, 1985a, 1985b).

It is interesting to interject that, in its early days, BASIC was considered to be a machine-independent computer language. Gottfried (1975) states, "Except for minor differences between one version of BASIC and another, the language is machine-independent. Hence a BASIC program can be run on many different computers" (p. 9). (It could, in fact, be argued that most computer languages are machine-independent during the first few years of use but become machine-dependent as they evolve and vendors begin adding features.)

As the numerous dialects of BASIC proliferated, several general criticisms emerged: (a) There were too many incompatible dialects, (b) Execution was very slow (due to BASIC's interpretive nature), (c) Variable names were often restricted to one or two characters, (d) BASIC lacked true subroutines (Kemeny & Kurtz, 1985a), (e) BASIC lacked local variables, and (f) BASIC lacked data structures (other than arrays). Due to these criticisms, BASIC's reputation became somewhat tarnished and its use was shunned by many.

In the early 1980s, Kemeny and Kurtz decided to rectify this confusing situation and began developing True BASIC. True BASIC not only adheres to the design criteria originally specified for BASIC, but also addresses many of the general criticisms mentioned above. True BASIC includes: (a) True subroutines; (b) Local variables; (c) Structured programming statements such as SELECT-CASE, DO WHILE, and DO UNTIL; and (d) Hardware-independent graphics.

The following discussion is divided into three areas: (a) Strengths, (b) Weaknesses related to the editor and function key definitions, and (c) Weaknesses in the language itself. These three categories will be addressed in turn.

Strengths of True BASIC

1. Elimination of the need for line numbers. Removal of required line numbering should make the language less bewildering to beginners and less intimidating to children

or to anyone who fears numbers and mathematics.

2. Addition of structured programming statements. The addition of structured programming statements such as IF-THEN-ELSE, WHILE LOOPS, external subroutines, and modules establishes True BASIC in the domain of general purpose programming languages such as COBOL, FORTRAN, and Pascal.
3. The design of True BASIC's loop structure. This structure can be represented as follows:

```
DO
   statement 1
   statement 2
   ...
   ...
   ...
   statement n
LOOP
```

   This syntax, which supports the WHILE and UNTIL conditional on either the DO or LOOP statement, is versatile and powerful. FORTRAN and versions of Street BASIC restrict the conditional to the statement beginning the loop. Pascal, on the other hand, has two loop statements, depending on whether the conditional WHILE or UNTIL is used. This may be confusing to beginning programmers.

   True BASIC also permits exits from the middle of a loop, an important feature that Pascal lacks. The authors of True BASIC provide a short comparison of True BASIC, Pascal, and FORTRAN's looping characteristics (Kemeny & Kurtz, 1985a). As they point out, the looping capabilities of True BASIC may be especially superior to those of Street BASIC and Pascal.
4. The ability to temporarily restrict editing changes to a section of a program or subroutine. Few, if any, editors provide this useful feature. Although every editor has some mechanism for specifying a range of lines to be affected, that range must be entered each time a command is directed at those specific lines. The omission, or incorrect entry, of the range can prevent intended changes from occurring or cause changes in inappropriate places. Such problems may be less likely to occur when using True BASIC. For example,

the temporary restriction feature allows multiple changes to lines 20 through 50 or to subroutine "X" without affecting other sections of the program.

5. The ability to execute a file of batch commands every time True BASIC is started. This allows the customizing of the True BASIC environment. For example, a teacher could create a file that would automatically cause a computer-assisted lesson to begin without the necessity of the user loading the file and then entering a start up command from the keyboard. The following lines (placed in a file called STARTUP.TRU) would load and begin execution of a sample grammar lesson (GRAMMAR is the filename of the True BASIC program containing the grammar lesson).

```
OLD GRAMMAR
RUN
```

6. The ability to define, save, and recall function key definitions. This allows the customizing of the keyboard while using True BASIC. For example, the following commands (entered at the True BASIC prompt) would define function key F11 to execute the current program and the key sequence Ctrl-P to enter the character string PRINT at the current cursor location (the Ctrl-A is used to terminate a definition).

```
KEY
  Press key to redefine: F11
  Define it as: RUN <cr> Ctrl-A


KEY
  Press key to redefine: Crtl-P
  Define is as: PRINT Ctrl-A
```

Redefined keys are in effect for the current True BASIC session only. The following command will save the new key definitions in a file called MYKEYS.

```
KEY TO MYKEYS
```

The following command will retrieve key definitions from a

file called MYKEYS.

```
KEY FROM MYKEYS
```

7. Descriptive and easy-to-use commands such as CHANGE, COPY, FIND, and MARK. This command simplicity is in marked contrast to counterpart commands in the vi editor of UNIX and edit_file of NOS/VE. Commands in these editors are enigmatic, highly abstract, and often require awkward keystroke combinations.
8. The use of a single key (INS on the IBM/PC) to toggle between insert and overstrike mode. This is a convenient feature not included in many editors.

Weaknesses in the Editor and Function Key Definitions

1. The definitions for some function keys (single or multiple key combinations used in lieu of commands to perform oft-repeated commands) are confusing and should be changed. Specifically, Ctrl Home (delete previous word), Ctrl PgUp (delete to end of current word), Esc (delete from cursor to start of line) and Ctrl End (delete from cursor to end of line) seem illogical and appear to have been an afterthought. Since True BASIC is not a word processor, the removal of Ctrl PgUp and Ctrl Home are recommended. They are not needed, since the deletion of a single word can be accomplished by holding down the delete key until all characters are removed. Larger blocks of text can be easily removed by using the mark text function key and deleting the block. Additionally, changing Esc to Alt-left-arrow and Ctrl End to Alt-right-arrow might be more appropriate for their intended functions.
2. The keystroke combination to pause program execution is awkward, and the combination to resume execution is susceptible to accidental triggering. True BASIC uses the key combination Ctrl/NumLock to cause an executing program to pause (allows the examination of output during program debugging). Any key can then be pressed to resume execution. This is probably a poor programming choice since an accidental key press can cause a premature resumption. Similar problems can occur when using any language or operating system. For example, the present authors have typed in the command DISKCOPY, paused to examine a reference manual, and accidentally allowed a corner of the manual to rest on the keyboard. The resulting keystroke

caused DISKCOPY to begin execution before the proper disks were in the drives. A solution to this True BASIC problem might be to use the common standard of Ctrl-S (pause) and Ctrl-Q (resume).

3. The OLD command is a poor choice for retrieving programs from disk. It appears the language developers chose OLD because it is the opposite of NEW (which clears the current program from the editor). Perhaps GET (the opposite of SAVE, which writes programs to disk) might be a better choice as it is more descriptive of what the command does. Two new function key definitions are suggested: (a) Ctrl HOME to supplant NEW, and (b) Alt HOME for recovering inadvertently cleared programs. Recovering cleared programs is a feature missing from True BASIC.
4. The command UNSAVE (used to erase a program from disk) is obscure. ERASE, which is used in many other applications, would probably be a better choice, since UNSAVE is not a real word.
5. The TRY and REPLACE commands might serve better as options on the CHANGE and SAVE commands. Options are more in line with Kemeny and Kurtz's original design criteria. Too many commands impose a burden on the beginning user.

A prime example of an excessive number of commands is Control Data Corporation's new operating system NOS/VE. The quick reference manual is over 1,200 pages in length and spans two physical volumes! There are so many commands that experienced systems analysts have difficulty remembering them.

Options, on the other hand, can be ignored. The system can (and should) provide default values. When a default is inappropriate, the system should prompt the user for the desired course of action. Possible alternatives might include PROCEED ANYWAY or ABORT THE COMMAND.

Weaknesses in the Language Itself

1. The NOLET option should be discontinued. The authors of True BASIC state that every line should begin with a command keyword (Kemeny & Kurtz, 1985a). Although compatibility with versions of Street BASIC is both important and desirable, NOLET is in direct conflict with the above concept. The removal of NOLET would enable the addition of continuation lines (discussed next).To promote the conversion of Street BASIC programs into True BASIC, a DO program might be advisable. Such a program would add LET

keywords in the appropriate places.

2. The language lacks continuation lines. Due to this, True BASIC statements longer than 80 columns extend past the edge of the screen. As the entire screen cannot be shifted to the right, it is difficult to view and modify such lines. If NOLET were removed, continuation lines could be implemented easily. Any line not beginning with a True BASIC keyword would automatically be a continuation of the previous line. If NOLET is kept, then some character sequence at the end of a line could indicate the presence of a continuation line. Perhaps a double period (..) or a double ampersand (&&) might be a good choice.

3. Radians are the default for trigonometric functions. Kemeny & Kurtz make the point that mathematicians use radians but the rest of us use degrees when computing angles, arcs, and other trigonometric figures (Kemeny & Kurtz, 1985c). Their choice of radians as the default and degrees as the option conflicts with their own design criteria and should be reversed.

4. The function MAXNUM is practical for finding the largest number, but EPS(0) for finding the smallest number is awkward and appears to violate several of the author's design criteria (specifically items 1 and 3 on the first page of this paper). This also appears to violate an unwritten rule the authors of True BASIC have adopted for naming commands-that of word/word-opposite (such as SAVE/UNSAVE, OLD/NEW). The addition of a function called MINNUM is recommended.

5. The SPLIT command is repeatedly issued to accommodate program output in the history window. This is a nuisance when switching between program execution and program editing. If SPLIT were permitted as a True BASIC statement, it would allow run-time control of the history and editing windows.

6. Circles and boxes are plotted relative to their lower left hand corner (Kemeny & Kurtz, 1985c, 1985b). While probably easier for the machine to plot, a center point with a radius (width and length for rectangles) would be more logical to programmers. Extra work should be done by the machine, not the user.

7. The collating sequences used in computer codes such as EBCDIC, ASCII, and CDC display code are arcane. The ASCII collating sequence of ABC...Z[\^-'abc...z used by most microcomputers makes it difficult to alphabetize names, addresses, and anything else containing both upper-case and

lower-case letters. Assuming the collating sequence will never be changed to AaBbCc...Zz (or aAbBcC...zZ), the addition of COLLATE= to the True BASIC OPTION statement is urged. COLLATE=UPPER would stipulate the normal ASCII collating sequence; COLLATE=LOWER, that the cases are reversed; COLLATE=UPPER/LOWER, that the lower-case letters are merged behind their upper-case counterparts; and COLLATE=LOWER/UPPER, that the upper-case letters are merged behind their lower case counterparts. Any other specification following COLLATE= would indicate the name of a file containing a user specified collating sequence. This would be a powerful addition to the language.

8. A PARAMETER statement should be added to True BASIC. While the value of variables can be modified during program execution, constants can not. A PARAMETER statement, such as that of FORTRAN, would allow symbolic names (variables) to be equated with a constant. Parameters ease the maintenance of constants used throughout a program. A single change in a PARAMETER statement propagates to all affected statements referencing the specified symbolic name (variable). Parameters are also useful in maintaining array dimensions, especially when several arrays each with multiple dimensions must be changed periodically (number of students in a class, questions on an exam).
9. True BASIC lacks the ability to inspect if a pixel is set or not set and check its color using the current screen coordinates. This ability would be beneficial for graphics applications involving animation when users may not want to draw over the top of an existing line or figure.
10. True BASIC lacks the ability to check the contents of any character position as well as its color. The addition of this feature would be helpful for graphics screens manipulating animated figures.
11. True BASIC lacks logical constants and variables. The addition of TRUE and FALSE values are recommended to make it easier when writing programs involving Boolean logic.
12. True BASIC lacks record structures. The addition of record structures would allow the creation and manipulation of data structures such as linked lists and binary trees. Their addition is highly recommended for the teaching of advanced programming concepts.

Summary

Overall, True BASIC is a well-designed language. It is easy to learn and use yet powerful enough for many sophisticated programming projects. Other educators will no doubt offer other enhancements. With such improvements, True BASIC may surpass both FORTRAN and Pascal as the language of choice, not only for teaching beginning programming, but also for use as a powerful general purpose programming language.

References

Davis, W. S. (1986). True BASIC primer. Reading, MA: Addison-Wesley.

Gottfried, B. S. (1975). Programming with BASIC. New York: McGraw-Hill.

Kemeny, J. G., & Kurtz, T. E. (1985a). Back to BASIC, the history, corruption, and future of the language. Reading, MA: Addison-Wesley.

Kemeny, J. G., & Kurtz, T. E. (1985b). True BASIC Reference Manual. Reading, MA: Addison-Wesley.

Kemeny, J. G., & Kurtz, T. E. (1985c). True BASIC Users Guide with Instructions for the IBM/PC. Reading, MA: Addison-Wesley.