An MBASIC® User Profile

R. L. Schwartz
DSN Data Systems Section

An important aspect of any programming language development is an awareness of how the language is actually being used. This article presents the preliminary results of an empirical study of MBASIC user programs. The study, part of an ongoing effort, seeks to discover the features and capabilities of the MBASIC processor which are being utilized. This will aid in future decisions to be made concerning language extensions and the development of a batch compiler.

I. Introduction

Designers of languages and compilers usually have comparatively little information about the way in which programming languages are actually used by typical programmers. Designers have some view of what programmers using the language do, but this is rarely based on a representative sample of the programs which are actually being written and used.

There has been widespread realization that more data about language use is needed. The modification of programming languages and compilers for programming languages should take place with a clear picture of what elements of the language are being used. Too often in the history of programming language and compiler development, decisions have been based on a feeling on the part of the language designers or compiler writers of what features are most important. It is easy to fall in the trap of assuming that complicated constructions are the norm when in fact they are infrequently used. Such mistaken assumptions can lead to misguided optimization efforts and misinterpretation of user needs.

Empirical studies of user programs written in the programming language FORTRAN by Knuth (Ref. 1) and in the language ALGOL 60 by Wichmann (Ref. 2) have been previously made. The studies have analyzed user programs in various ways. A *static* profile provided information about the frequency of occurrence of various statements and constructs, complexity of control flow, as well as other data which can be gathered from a program listing. A *dynamic* profile gave information related to frequency of execution of program statements, and how the execution time was distributed among the program statements. The overall conclusions of the studies showed that the vast majority of the statements in most programs are of a very simple nature, and that the bulk of the program execution time is spent executing a very small percentage of the program statements.

Such a profile of typical programs written in the MBASIC computer programming language has been initiated at JPL. The MBASIC language is an advanced version of BASIC designed at the Laboratory in the early part of 1971. Presently an MBASIC interpreter is implemented on the UNIVAC 1108 and DECsystem-10 computers. It has recently been recommended as the standard nonreal-time language for use in Deep Space Network program development. Intended to be a simple easy-to-learn general-purpose programming language, it is used extensively throughout the Laboratory for management, scientific, and light computing applications. References 3 and 4 give a general introduction to the features and capabilities of the MBASIC system.

II. The Study Performed

A preliminary *static* profile of MBASIC user programs analyzing a total of 114 programs stored in the MBASIC user library has been made, with programs chosen randomly from approximately 800 user program files. The statistics described in the following sections attempt to analyze the use of different statement types, modifiers, and the use of the backslash statement regenerator. These statistics were compiled automatically by a computer program written in the MBASIC language. Direct analysis of selected programs was used to aid in the interpretation of the results.

A. Overall Statement Usage

A top-level statement usage breakdown of the 14,908 statements appearing in the 114 MBASIC user files is given in Table 1. This statement breakdown does not include statements embedded in an IFTHENELSE statement (e.g., the GOTO statement embedded in IF B THEN GOTO 100). Statistics for this were compiled separately, and are reported in the next section. In the case where there are synonymous keywords (i.e., PRINT/WRITE, and STOP/END), the combined statistics are reported with the individual statistics given in a footnote. The assignment statement is somewhat of a special case: it may be initiated by a LET keyword for one or more simple assignments, by an EXCHANGE keyword for one or more variable exchanges, or without a keyword for a combination of assignments and exchanges. Each of these cases is shown separately in Tables 1 to 5.

A special analysis was made for the GOTO and GOSUB statements. The specification of the line number for the branching operation can, in general, be an arbitrary expression evaluating to a real number. A check was made to see how often a simple constant line number was used rather than an expression. This is also reported in Table 1.

B. IFTHENELSE Statement Usage

Tables 2 and 3 show the frequency of occurrence of the different program statements in the THEN and ELSE clauses of the IFTHENELSE statement, respectively. The preliminary results presented herein only analyze *one nesting level*. Thus in a statement such as

```
IF D
  THEN
    IF C
      THEN A = 14
      ELSE B = 7
  ELSE PRINT A
```

the assignments to the variables A and B would not be recorded. The statement in the THEN-clause would be noted as an IF statement, and the statement in the ELSE-clause would be noted as a PRINT statement. Future empirical studies may analyze arbitrary IFTHENELSE nesting levels.

C. Modifier Usage

Table 4 summarizes the results concerning the use of the WHERE, IF, UNLESS, and FOR modifiers in program statements. Again, only a preliminary analysis of the top-level statements was performed (e.g., the WAIT statement in IF B THEN WAIT UNLESS X is not analyzed for modifier usage).

D. Backslash Usage

Table 5 reports the results concerning the use of the backslash statement regenerator for each of the statement types. Again, only the top-level statements have been analyzed.

III. Analysis of Results

Before analyzing the results given in Tables 1 to 5, the reader should note that Table 1, giving the statement usage breakdown, may tend to present a slightly misleading view of the relative usage of statements in programs. One might register surprise (or even joy) at the fact that the GOSUB statement is used more frequently than the GOTO statement. While this is indeed true at the top-level of statement nesting, the table does not show the extremely heavy usage of GOTO statements in IFTHENELSE statements. When the figures in Table 1 are modified to include the second level statements in the THEN and ELSE clauses of the IF statement, the results then show: ASSIGN 28.3% (4975), IF 12.8% (2243), GOTO 12.6% (2213), PRINT/WRITE 12.5% (2193), and GOSUB 7.5% (1325). Thus, while the GOSUB is used more frequently as an unconditional statement, the GOTO is used more overall. This breakdown is in contrast to the results reported by Knuth concerning FORTRAN programs written at Lockheed. He found the following statements most heavily used: Assignment 41%, IF 14.5%, GOTO 13%, CALL 8%, and WRITE 4%. The non-numeric applications for typical MBASIC programs would seem to account for the higher frequency of output statements. Generally, however, the frequency of usage of MBASIC statements follows the usage of FORTRAN statements rather closely.

The APPEND, PAUSE, REMOVE, DATA, WIDTH, READ, RENAME, EXIT and RANDOMIZE statements were very seldom used, while the EXCHANGE, TAPE, and REMARK statements were never used in the samples studied. The relatively high position for the LET statement (0.5%) was found to be due to one program which was translated from the BASIC language.

Table 2 shows that almost half of all THEN clauses in IF statements were GOTO statements, with an additional one quarter being assignment statements. The use of the GOTO, assignment, and PRINT/WRITE statements accounted for 82% of all THEN clause statements. Multiple levels of nesting of the IF statement in the THEN clause accounted for about 6% of the uses of the IF.

The use of the ELSE clause of the IF statement was somewhat different from that of the THEN clause. Assignment statements accounted for about a third of all ELSE clauses, with the GOTO accounting for about another quarter, the multi-level IF structures nested in the ELSE clause accounting for about another fifth. The high percentage of nested IF statements is probably due to the lack of a CASE-type construct in the MBASIC language.

The 46% GOTO and 26% ASSIGN for THEN clauses compared to 32% ASSIGN and 25% GOTO for the ELSE clauses can probably be explained in terms of a lack of ability to associate more than one statement with a THEN or ELSE clause. Multi-statement THEN and ELSE clauses are simulated by having a sequence of the form:

```
ℓ        IF B THEN GOTO ℓ + i
ℓ + 1    S1-ELSE      !ELSE CLAUSE
            .
            .
            .
ℓ + i - 1 GOTO ℓ + j  !BRANCH OUT OF IFTHENELSE
ℓ + i    S1-THEN      !THEN CLAUSE
            .
            .
            .
ℓ + j    !END OF IFTHENELSE
```

Thus the ELSE clause which appeared 43% of the time that the IF statement was used, is only used for single statement clauses. This statement was most frequently an assignment statement (32% of the time).

The modifier usage, given in Table 4, showed a rather limited use of modifiers. The UNLESS modifier was infrequently used, with only six statement types making any use of the modifier (approximately 50% of that usage was with the GOSUB statement). The highest percentage of statements with modifiers employed is the GOSUB statement, with about half of the GOSUB statements using modifiers. Of these statements, about 72% were WHERE modifiers. This is due to a lack of a procedure mechanism that allows the passing of parameters. This has been simulated with a statement of the form:

```
GOSUB line WHERE PARM1 = vaℓ1, PARM2 = vaℓ2 . . .
```

For procedures that return values (functions), the passing of the return value has been simulated with a statement of the form:

```
RETURN WHERE ANSWER = val
```

This was done about 12% of the time the RETURN statement was used.

The use of the FOR modifier is confined mostly to input/output statements and the assignment statement, with 67% of the FOR modifiers being used for the PRINT/WRITE and INPUT statements, and 30% being used for initialization of arrays in assignment statements. It is interesting to note that a GOTO statement was used with a FOR modifier three times (although an explanation of this use cannot be given here).

The IF modifier had 93% of its usage confined to three types of statements: 33% to branching statements (GOTO, and GOSUB), 28% to assignment statements, and 32% to input/output statements (PRINT/WRITE, and INPUT).

The use of the backslash, illustrated by Table 5, is confined to only 5 statement types. Approximately 36% of the PRINT/WRITE statements used a backslash. This accounted for 71% of the backslash usage, with an additional 21% of the backslashes being used for assignment statements (to limit the scope of modifiers).

Of the two pairs of synonymous keywords, PRINT/WRITE and STOP/END, both showed heavy favorites. PRINT was used about 84% of the time instead of WRITE, while END was used 79% of the time instead of STOP.

IV. Conclusions

Depending on the type of information desired, there are many different ways the data presented in Tables 1 to 5 could have been analyzed. Further efforts are needed to arrive at a more comprehensive evaluation of the data.

Overall, the data suggests that the fancier or more complex features of the MBASIC language are not being utilized extensively. The statistics lend credence to the hypothesis that the users of the MBASIC language are attempting to simulate some of the elementary structured programming constructs such as procedures with parameters, an IFTHENELSE statement with multi-statement THEN and ELSE clauses, and a CASE statement.

Users have attempted to introduce techniques for greater abstraction and problem reduction by simulating procedures with parameters, and using non-numeric GOSUB expressions. Some 38% of all GOSUB line expressions have used meaningful variable names to serve as procedure names (with the variable initialized to the proper line number elsewhere). These conclusions suggest that the MBASIC structured programming extensions will eliminate much of the present use of modifiers.

MBASIC users have not taken advantage of the ability to use modifiers and backslashes as a general language feature; rather, its use has been confined mainly to input/output statements and assignment statements, places where other languages have allowed a similar capability. The exception to this is the use of the WHERE modifier, but in that case it was used to circumvent the lack of adequate abstraction mechanisms.

V. Future Work

This article has presented only a preliminary look at what needs to be an ongoing effort. Only a limited static profile has thus far been performed.

Future work should take place in three areas: a more comprehensive static profile, the development of a dynamic profile, and, at a later time, the integration of such information into the design considerations attendant to the planned MBASIC batch compiler.

The static profile of MBASIC programs can be enhanced in many ways. A larger sample of user programs would certainly contribute to more representative data. The automatic measurement program should be expanded to analyze the use of arbitrary IFTHENELSE nesting levels, the composition of assignment statements and FOR-NEXT loops (as was done for FORTRAN and ALGOL 60), and spacial characteristics of programs. In addition, these statistics should also report on the mean usage (with variances) of each of the language features in individual programs.

A dynamic profile can be developed, which gives information about frequency of execution of statements and the distribution of the execution time over the program statements. A more ambitious, and important, study would determine if users are taking advantage of the dynamic scope rules of the MBASIC language. As the language presently exists only in interpretive form, with incremental parsing, the interstatement relationships, such as the association between a FOR statement and the corresponding NEXT statement, must be determined on the basis of execution order rather than textual order. An important question is whether MBASIC language users are using this dynamic association rule. This has important consequences for the development of an MBASIC compiler, since it is intended that most MBASIC programs shall function exactly the same in both interpretive and compiled modes.

A long range goal is to incorporate, at user option, such dynamic and static profiles into the batch monitor, as part of the user program development cycle. Static information obtained during compile time, and dynamic information obtained at run time can be used to direct program optimization during the next compilation. Thus, the compiler effort can be concentrated on those portions of the code which are most important to program function. Conceptually, this information could be used, via a feedback loop between the execution phase and compile phase of program development, to have the program execution history direct a heuristic optimization of the object code produced by the compiler.

References

1. Knuth, D. E., "An Empirical Study of FORTRAN Programs," *Software Practice and Experience*, Vol. 1, pp. 105-133, 1971.

2. Wichmann, B., *Algol 60 Compilation and Assessment,* Academic Press, N. Y., 1973.

3. *MBASIC Fundamentals, Vol. 1,* Feb. 1974 (JPL internal document).

4. Schwartz, R., *Syntactic Description of the MBASIC Language,* October 1976 (JPL internal document).

Table 1. Top-level statement usage breakdown

Type	% of total (actual No.)	Type	% of total (actual No.)
ASSIGN	28.2 (4209)	STOP/END	0.5 (79)
IF	13 (1944)	LET	0.5 (74)
PRINT/WRITE	12.9 (1928)	COPY	0.4 (57)
GOSUB	8.2 (1227)	ON	0.3 (43)
GOTO	7.5 (1112)	APPEND	0.2 (26)
INPUT	6.7 (999)	PAUSE	0.2 (24)
RETURN	3.7 (553)	REMOVE	0.2 (23)
FOR	3.5 (528)	DATA	0.1 (21)
NEXT	3.5 (527)	WIDTH	0.1 (8)
STRING	3.3 (496)	READ	0.1 (8)
OPEN	1.9 (276)	RENAME	0 (5)
CLOSE	1.9 (229)	EXIT	0 (3)
DIM	1.5 (223)	RNDMZ	0 (1)
REAL	0.7 (101)	EXCHANGE	0 (0)
AT	0.7 (100)	TAPE	0 (0)
WAIT	0.6 (84)	REMARK	0 (0)

File count: 114
Statement count: 14,908

GOTO with constant line expression: 97% (1075)
GOSUB with constant line expression: 62% (756)

PRINT	10.8%	(1613)
WRITE	2.1%	(315)
STOP	0.1%	(17)
END	0.4%	(62)

Table 2. THEN clause statement usage breakdown

Type	% Total THEN clauses (actual No.)	Type	% Total THEN clauses (actual No.)
GOTO	45.8 (891)	TAPE	0 (0)
ASSIGN	25.8 (501)	AT	0 (0)
PRINT/WRITE	10 (204)	DATA	0 (0)
IF	6.2 (121)	NEXT	0 (0)
RETURN	4.5 (87)	APPEND	0 (0)
GOSUB	4 (77)	OPEN	0 (0)
INPUT	1.6 (31)	REAL	0 (0)
END/STOP	0.7 (13)	REMARK	0 (0)
FOR	0.5 (9)	EXIT	0 (0)
COPY	0.3 (6)	READ	0 (0)
ON	0.1 (2)	STRING	0 (0)
REMOVE	0.1 (1)	WIDTH	0 (0)
CLOSE	0.1 (1)	EXCHANGE	0 (0)
LET	0 (0)	RNDMZ	0 (0)
PAUSE	0 (0)	DIM	0 (0)
RENAME	0 (0)	WAIT	0 (0)

THEN Clause count: 1944

PRINT	7.8%	(152)
WRITE	2.7%	(52)
END	0.6%	(12)
STOP	0.1%	(1)

Table 3. ELSE clause statement usage breakdown

Type	% Total ELSE clause (actual No.)	Type	% Total ELSE clauses (actual No.)
ASSIGN	31.9 (265)	PAUSE	0 (0)
GOTO	25.3 (210)	RENAME	0 (0)
IF	21.4 (178)	TAPE	0 (0)
PRINT/WRITE	9 (75)	AT	0 (0)
GOSUB	4 (21)	DATA	0 (0)
INPUT	3.7 (31)	NEXT	0 (0)
RETURN	3 (25)	WAIT	0 (0)
ON	0.8 (7)	OPEN	0 (0)
COPY	0.8 (7)	REAL	0 (0)
REMOVE	0.5 (4)	REMARK	0 (0)
FOR	0.4 (3)	EXIT	0 (0)
CLOSE	0.2 (2)	READ	0 (0)
STOP/END	0.2 (2)	STRING	0 (0)
APPEND	0.1 (1)	WIDTH	0 (0)
DIM	0 (0)	EXCHANGE	0 (0)
LET	0 (0)	RNDMZ	0 (0)

If statements with ELSE Clause: 43% (831)

PRINT	7.3% (61)
WRITE	1.7% (14)
STOP	0.1% (1)
END	0.1% (1)

Table 4. Modifier usage by statement type

Type	% Using modifiers (actual No.)	% Total WHERE	% Total IF	% Total UNLESS	% Total FOR
GOSUB	51.2 (628)	72.5	13	13.5	.9
WAIT	40.5 (34)	0	0	100	0
GOTO	32.3 (359)	12.4	86.5	.3	.8
WRITE/PRINT	30.9 (596)	1.5	36.6	3.6	58.3
APPEND	30.8 (8)	0	25	62.5	12.5
WIDTH	25 (2)	0	100	0	0
INPUT	24.1 (241)	2.3	35.5	0	62.2
FOR	21.8 (115)	99.1	.9	0	0
REMOVE	21.7 (5)	0	80	0	20
READ	12.5 (1)	0	0	0	100
COPY	12.3 (7)	0	71.4	0	28.6
RETURN	11.8 (65)	66.2	33.8	0	0
ASSIGN	11.7 (493)	8.6	47.3	4.7	39.3
STRING	10.3 (51)	55.8	23.4	0	20.8
CLOSE	10 (23)	84	12	0	4
REAL	9.9 (10)	58.3	41.7	0	0
STOP/END	8.9 (7)	14.3	85.7	0	0
OPEN	8.3 (23)	0	48.3	0	51.7
PAUSE	4.2 (1)	100	0	0	0
ON	2.3 (1)	0	100	0	0
AT	2 (2)	0	100	0	0
DIM	1.8 (4)	0	100	0	0
NEXT	.6 (3)	100	0	0	0
RNDMZ	0 (0)	–	–	–	–
TAPE	0 (0)	–	–	–	–
EXCHANGE	0 (0)	–	–	–	–
LET	0 (0)	–	–	–	–
RENAME	0 (0)	–	–	–	–

WRITE	50.5%	1.1%	47%	0%	51.9%
PRINT	27.1%	1.7%	31.9%	5.2%	61.1%
END	6.5%	0%	100%	0%	0%
STOP	17.6%	33.3%	66.7%	0%	0%

Table 5. Backslash usage by statement type

Type	% Using backslash (actual No.)	Type	% Using backslash (actual No.)
PRINT/WRITE	35.9 (693)	TAPE	0 (0)
STOP/END	8.9 (7)	AT	0 (0)
ASSIGN	6.1 (256)	CLOSE	0 (0)
INPUT	1.7 (17)	DIM	0 (0)
GOSUB	.2 (2)	EXCHANGE	0 (0)
APPEND	0 (0)	GOTO	0 (0)
COPY	0 (0)	LET	0 (0)
NEXT	0 (0)	OPEN	0 (0)
PAUSE	0 (0)	READ	0 (0)
REAL	0 (0)	RENAME	0 (0)
RNDMZ	0 (0)	WIDTH	0 (0)
REMOVE	0 (0)	WAIT	0 (0)
STRING	0 (0)		

PRINT	37%	(597)
WRITE	30.5%	(96)
STOP	11.8%	(2)
END	8.1%	(5)