Comput. Educ. Vol. 6, pp. 179 to 192, 1982
Printed in Great Britain

0360-1315/82/020179-14$03.00/0
Pergamon Press Ltd

COMAL 80—ADDING STRUCTURE TO BASIC

M. A. Bramer
Mathematics Faculty, The Open University, Milton Keynes, MK7 6AA, England

(*Received* 27 *January* 1981; *amended* 28 *April* 1981)

Abstract—BASIC has developed almost to the status of a standard language in many areas of educational computing, despite strong criticism by theoreticians and considerable weaknesses (in particular, the lack of facilities for "structured programming") which are handicaps when writing all but the most elementary programs.

Some of the features of BASIC which have led to its considerable popularity are examined. These features together with the widespread use of BASIC suggest that if a more appropriate introductory programming language is to be developed and accepted it must build on rather than replace BASIC.

A programming language is described which was designed to remedy the weaknesses of BASIC by adding structured constructs and other desirable improvements, whilst retaining its major features and overall spirit.

This language—known as COMAL 80—was developed in Denmark and is proposed as a significant yet realistic improvement on BASIC as a programming language for schools.

"Marley was dead, to begin with. There is no doubt whatever about that. The register of his burial was signed by the clergyman, the clerk, the undertaker, and the chief mourner. Scrooge signed it.... Old Marley was as dead as a door-nail.... There is no doubt that Marley was dead. This must be distinctly understood." (Charles Dickens—A Christmas Carol)

"Reports of my death are greatly exaggerated." [Mark Twain (in a cable to the Associated Press)]

The two quotations reproduced above come naturally to mind when considering the imminent demise of the BASIC programming language and its replacement by ALGOL-68 (or Pascal, PL/I or whatever), a favourite topic of conversation in academic computer science circles.

It is notable that the most popular languages—BASIC, FORTRAN and COBOL—are also the most widely and vehemently criticised. Although frequently pronounced moribund, in reality all three are flourishing and continuing to evolve.

BASIC (the name stands for Beginners' All-purpose Symbolic Instruction Code) was developed at Dartmouth College, U.S.A. in the mid-1960s by John Kemeny and Thomas Kurtz, principally as a language for use by novice students working in an interactive mode. In the intervening years, countless more powerful or "sophisticated" languages have come and gone almost unnoticed, whilst BASIC has gained worldwide popularity, reaching the status of a *de facto* standard language in many areas of educational computing (especially school computing) as well as for "personal" or hobbyist computing. The great majority of criticism of BASIC in the educational world not only fails to explain the reason for its considerable popular success but unrealistically ignores the huge commitment, in terms of personal effort and practical experience as well as of software development, which the users of a language have invested in its use. For most practising teachers, it is simply not practical to abandon the use of BASIC whatever the theoretical deficiencies which others may ascribe to it. The language is widely available, well-known, there are numerous introductory textbooks and an ever-increasing volume of educational software. These are not advantages to be lightly discarded. In this article, it is argued that even leaving aside these advantages there are good practical reasons for the popularity of BASIC as a language for beginners and non-computing specialists, but that there are identifiable weaknesses which are handicaps when writing all but the most elementary programs. A programming language designed to remedy these weaknesses whilst retaining the major features and overall spirit of BASIC is COMAL 80 (COMmon ALgorithmic language), which was developed in Denmark, and this is proposed as a significant yet realistic improvement on BASIC as a programming language for schools.

In the absence of an agreed "standard" version of BASIC*, examples will be based on the version originally implemented on the Hewlett-Packard 2000 series of minicomputers, since this is both reasonably typical and well-known in educational circles. It is recognised that other versions may be considerably different in particular features.

It is assumed that the reader is generally familiar with BASIC.

Before describing the features of COMAL it is worthwhile digressing to examine some of the characteristics of BASIC which may account for its popularity, since any argument for improving BASIC begs the question why it should be retained at all.

WHY IS BASIC SO POPULAR?

BASIC is oriented towards use at a terminal, in an interactive mode, by the beginner (and the poor typist). By contrast, many other languages were originally designed for use in batch mode with all input on 80-column punch cards and all output to a 132-character width line printer. Even when used from a terminal, such languages frequently reflect their age, for example by providing no facilities for interactive input of data.

Some particular features of BASIC are as follows. (Comparisons are drawn with other languages where appropriate.)

Typing aids

(a) Lines are input in "free-format", with spaces not significant. (c.f. the precise and error-prone spacing needed for FORTRAN programs, with 6 initial spaces for "normal" statements, then up to 66 characters, followed by up to eight characters for comments.)

(b) Program statements can be entered in any order and are automatically placed in the correct sequence.

Debugging aids

(a) Syntax checking takes place on a line-by-line basis as program statements are typed, thus it is not possible to store or list an invalid line of BASIC. Only a small number of multi-line syntax errors (e.g. "unmatched FOR and NEXT") are not detected this way, and these are easy to find when a program is RUN. A syntax error in one line cannot "contaminate" the analysis of any subsequent lines to produce incorrect error messages, a common feature of many compilers.

(b) Run-time errors identify the offending statement directly by means of its line number (c.f. the cryptic error messages or error numbers produced by many systems).

(c) Facilities for tracing program execution and "dumping" the values of variables are often available.

Ease of use

(a) It is extremely easy to "get started" in BASIC. Having logged-on it is only necessary to type

```
10 PRINT "HELLO"
20 END
RUN
```

Most other languages (and their implementations) perform badly on this test.

(b) There is no need for the beginner to learn how to use a text editor to input or change programs, since line numbers are used to replace and delete program statements as well as to sequence them.

(c) There is no need to learn a separate Operating System command language to manipulate programs and files or to specify input or output devices. BASIC has its own set of simple commands (LIST, RUN, SAVE etc.) which refer in an obvious way to the user's "current" program etc. With many systems the user is either automatically placed inside the BASIC operating environment or can enter it easily by typing one or two prescribed lines. In such a "dedicated" system, it is not possible to obtain an error message which refers to routines or facilities outside the BASIC system—there are none. Such messages are a considerable source of difficulty with many other systems. (Additional facilities available to privileged users of a dedicated system, such as the system operator, will generally take an identical format to the BASIC commands available to all.)

* An official standard has recently (and belatedly) appeared, but has made little impact as yet.

Interactive features

(a) Input and output are normally at the terminal and the user is normally 'prompted' for input to be provided for a running program. (By comparison, both Pascal and ALGOL-68 experience difficulties when the same file—e.g. the terminal—needs to be open for both input and output simultaneously.)

(b) At the end of program execution, control returns to the terminal and the program remains unaltered in the user's "workspace". It can thus be changed and rerun in a simple fashion, or the values of specific variables can be "dumped". Some BASIC systems allow particular BASIC statements (e.g. LET X = 5) to be executed in "immediate mode", i.e. as commands, followed by restarting the program.

Although the features listed above comprise genuine advantages over its competitors, it is clear that they do not relate to the BASIC language as such, but rather to the *operating environment* (frequently a dedicated environment) in which it is generally used. The normal use of an interpreter rather than a compiler for BASIC makes many of the user aids described above much easier to provide, and the lack of compiled code is not a significant problem for most reasonably small programs, although it can become so as BASIC is used for more and more ambitious projects. The use of line numbers for program input and editing is an especially valuable feature. Naturally, a BASIC-like environment could be built around most (perhaps all) programming languages, but perhaps surprisingly this has not taken place.

Turning to the BASIC language itself, its most helpful features, especially for the beginner, probably lie in the avoidance of many of the sources of difficulty of its more sophisticated rivals*.

(a) No "awkward" syntax
c.f. quotation marks around keywords, such as "BEGIN" in ICL ALGOL-60
.GT. etc. in FORTRAN
different meaning for round and square brackets (ALGOL-68)
multiple parentheses, such as (CAR(CDR(CDR(CDR(X)))) in LISP
parentheses in IF statements, e.g. IF (A.EQ.B) GOTO 20, in FORTRAN.
(b) No explicit declaration of simple variables.
(c) No complex syntactic constructions, e.g. REF REF INT in ALGOL-68.
(d) No unnatural rules, e.g. "right to left" application of operators in APL (e.g. 4 − 2 − 2 = 4).
(e) Few rules for ordering of statements.
(f) No "obscure" rules, such as for FORTRAN "format" statements.
(g) Simple statements for terminal input and output.

On the positive side, the string handling facilities of BASIC, where a string is regarded as a variable length quantity (not, for example, as an array of characters) are arguably superior to those of any other widely used language, with the exception of SNOBOL, which is specifically oriented towards string processing applications.

It is worthwhile to point out that many of the advantages of BASIC could equally well be viewed as disadvantages, for example the lack of declarations makes certain program errors (e.g. mistyping 10 LET X = X + 1 as 10 LET Y = X + 1) extremely difficult to detect. However for the purposes for which it was originally intended the advantages of simplicity are probably decisive. The strengths and weaknesses of BASIC are discussed more fully in [5].

BASIC gains most by comparison with the typical mode of use of other languages on a mainframe or mini-computer system: input the source program into a file (with no syntax checking), invoke a compiler to read in the source file and find syntax errors, use a text editor to change the source program, exit from the editor, reinvoke the compiler to find any further errors, and so on. With the rapid introduction of microcomputer systems aimed at the inexperienced or isolated user, the provision of user aids and debugging features is becoming a major design aim rather than a distracting afterthought. Such facilities will increasingly be developed—doubtless in improved form from those described above—and eventually be made available with *every* language. In particular, the development of easy-to-use screen-oriented editors provides an effective alternative to BASIC's use of line numbers for editing purposes.

Naturally BASIC will also benefit from the growing emphasis on user aids, the Sinclair ZX80 feature of one keystroke per keyword being a simple example of this, but it is inevitable that the

* Although some of the features listed below have no justification, in many cases they are unavoidable concomitants of the sophistication of the languages. The analysis here is given from the viewpoint of the beginner, not the experienced user.

pre-eminent position of BASIC as a straightforward and strongly user-oriented language will be severely undercut. A promising start in this direction is the UCSD Pascal system developed by Kenneth Bowles at the University of California at San Diego in the United States[2]. UCSD Pascal has been well received in educational circles in the United States and has started to gain popularity in Britain, although it is not yet in wide use here. Whether the language will ultimately suffer the same fate as PL/I, ALGOL-68 and the rest remains to be seen. Pascal itself has been in existence since 1968 and made little impact prior to the development of the UCSD system.

The practical difficulties associated with abandoning an established language (even one, such as FORTRAN, with severe and obvious deficiencies) make a powerful argument against such "revolutionary" change.

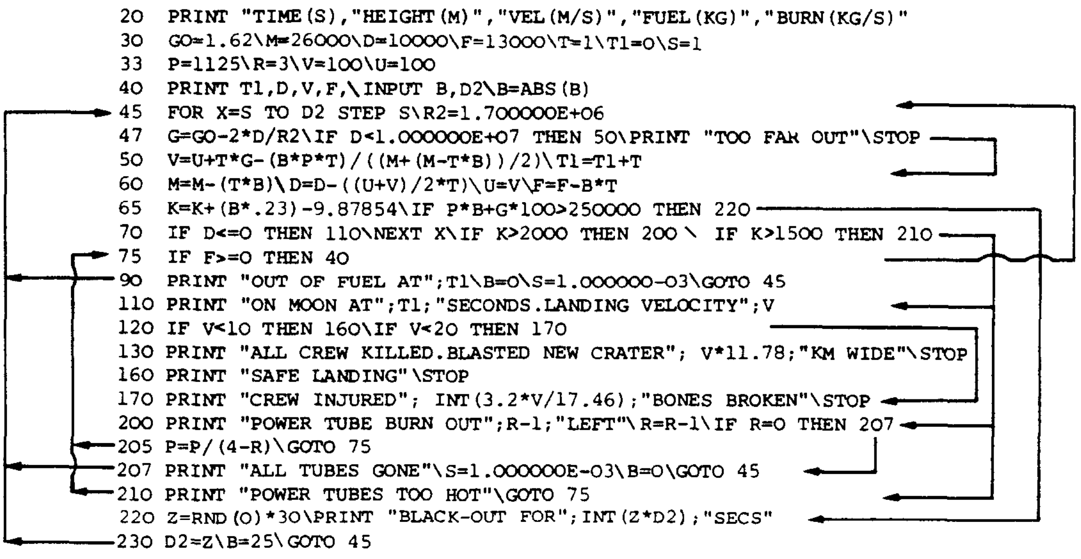
The alternative option of an evolutionary approach, i.e. improving BASIC rather than replacing it, is an attractive one. However in choosing this option it is essential not to lose sight of the original objectives of the language, as an easy to learn, easy to use, straight-forward language (or system) aimed particularly at the beginner and non-computer specialist. Many of the available extended forms of BASIC seem to have concentrated on providing more and more powerful features for advanced programming. However, to extend BASIC to have the power of (say) PL/I would be a serious mistake.

The designers of the set of extensions embodied in the programming language COMAL have avoided this pitfall and have rightly concentrated on removing the major source of unnecessary complexity in BASIC programming—and incidentally an area in which it is much inferior to UCSD Pascal—the lack of facilities for structured programming.

WHY IS BASIC UNPOPULAR?

Apart from the notorious use of "short" (i.e. one or two character) names for variables, the most significant criticism of BASIC is that made by the advocates of "structured programming". It is now widely agreed that the unrestricted use of "GOTO" statements (including conditional jump statements such as 20 IF X<3 THEN 30) is an undesirable practice which tends to lead to the development of programs which are extremely hard to read, modify or debug.

Unfortunately, BASIC—like most other languages developed before the principles of structured programming became accepted—positively encourages such uncontrolled use of GOTO statements, and the common practice of debugging (or even writing) programs on-line tends to aggravate this problem further. A striking example is given by Atherton[1] of a "moon landing" program of only about 20 lines, but which it seems virtually impossible to comprehend. The program is reproduced below as Fig. 1, the lines on the left and right hand sides denoting unconditional and conditional transfers of control, respectively. Although a particularly poor example of BASIC programming (not



```
20  PRINT "TIME(S),"HEIGHT(M)","VEL(M/S)","FUEL(KG)","BURN(KG/S)"
30  GO=1.62\M=26000\D=10000\F=13000\T=1\T1=0\S=1
33  P=1125\R=3\V=100\U=100
40  PRINT T1,D,V,F,\INPUT B,D2\B=ABS(B)
45  FOR X=S TO D2 STEP S\R2=1.700000E+06
47  G=GO-2*D/R2\IF D<1.000000E+07 THEN 50\PRINT "TOO FAR OUT"\STOP
50  V=U+T*G-(B*P*T)/((M+(M-T*B))/2)\T1=T1+T
60  M=M-(T*B)\D=D-((U+V)/2*T)\U=V\F=F-B*T
65  K=K+(B*.23)-9.87854\IF P*B+G*100>250000 THEN 220
70  IF D<=0 THEN 110\NEXT X\IF K>2000 THEN 200\ IF K>1500 THEN 210
75  IF F>=0 THEN 40
90  PRINT "OUT OF FUEL AT";T1\B=0\S=1.000000-03\GOTO 45
110 PRINT "ON MOON AT";T1;"SECONDS.LANDING VELOCITY";V
120 IF V<10 THEN 160\IF V<20 THEN 170
130 PRINT "ALL CREW KILLED.BLASTED NEW CRATER"; V*11.78;"KM WIDE"\STOP
160 PRINT "SAFE LANDING"\STOP
170 PRINT "CREW INJURED"; INT(3.2*V/17.46);"BONES BROKEN"\STOP
200 PRINT "POWER TUBE BURN OUT";R-1;"LEFT"\R=R-1\IF R=0 THEN 207
205 P=P/(4-R)\GOTO 75
207 PRINT "ALL TUBES GONE"\S=1.000000E-03\B=0\GOTO 45
210 PRINT "POWER TUBES TOO HOT"\GOTO 75
220 Z=RND(0)*30\PRINT "BLACK-OUT FOR";INT(Z*D2);"SECS"
230 D2=Z\B=25\GOTO 45
```

Fig. 1. Moon landing program.

least because of the multiple statements appearing on most lines). the program epitomizes the difficulties which arise in understanding the convoluted flow of control of all but the smallest BASIC programs. The term "spaghetti-like control paths" has been used to describe this phenomenon.

It is easy to guess that the original program logic has been extended piecemeal until it is now no longer intelligible. If in running the program an error were reported at line 50, say, it would be extremely difficult to determine the path by which the program's "flow of control" had come to that point.

A further weakness of BASIC is the poor form of subroutine facility provided (GOSUB and RETURN statements). Such "open subroutines" have two major disadvantages: they are "unnamed", with no well-defined entry or exit point (there is nothing to prevent a GOSUB statement which refers to a line which is not the start of a subroutine, by accident) and there is no parameter passing mechanism. The use of such inadequate facilities is also likely to impede an understanding of the value and significant features of conventional (i.e. FORTRAN-like or ALGOL-like) subroutines.

COMAL/STRUCTURED BASIC

The programming language COMAL was developed in Denmark by Borge Christensen (State Teachers' College, Tonder) and Benedict Løfstedt (Department of Computer Science, University of Aarhus) as a set of extensions to BASIC aimed at providing facilities for structured programming and remedying other observed defects. The name COMAL stands for COMmon ALgorithmic language, but the alternative title "Structured BASIC" is sometimes used and is far more indicative of the nature of the language. The language was intended primarily for use in state schools and has been used successfully at that level in Denmark since 1975 and is steadily gaining in popularity. A number of implementations of COMAL, on a variety of mini- and microcomputer systems, exist in Denmark. At the time of writing, three of these are available in Britain: those for the Digital Data Electronics SPC-1, the RC702 Piccolo and the Commodore PET 8032 microcomputers. A number of other manufacturers well-known in the educational field are known to be interested in implementing COMAL in Britain and it is likely that several other versions will soon be available.

Inevitably a number of different extended versions of COMAL have been implemented in Denmark since 1975. A specification of a standardized (and improved) version has recently been produced by a group of manufacturers, computer scientists and educationalists and this is known as COMAL 80.

COMAL extends BASIC by adding features taken from other languages, notably Pascal, whilst remaining recognisably BASIC. Programs written in BASIC will still run successfully in COMAL, either directly or with only slight modifications (such as would be necessary in transferring to another slightly different version of BASIC). Changing from BASIC to COMAL is thus a practical and relatively straightforward step to take, with the use of the structured programming features being introduced into teaching, perhaps in a gradual fashion.

For the newcomer, it is probably best to introduce the "structured" style of programming at the outset, where possible, since experience shows that a bad programming style, once gained, can be difficult to remedy later.

The following description relates to the final proposal for the nucleus of the COMAL 80 language[4]. Earlier versions of COMAL naturally differ from this in some respects. Several of the examples given below are adapted from those in[3].

A set of recommendations for standard extensions to the language nucleus (including file handling facilities) is currently in preparation.

From the examples given below, it will be noted that there are a number of places (notably the assignment statement) where there are slight differences between the syntax of COMAL 80 and that of BASIC. In practice, it is likely that most implementations of COMAL 80 will provide the traditional BASIC forms as an alternative.

In comparing COMAL with BASIC below, the conventional (i.e. Dartmouth College or Hewlett-Packard) form of BASIC has been used as the basis of comparison. It should be pointed out that a number of recent versions of BASIC (especially microcomputer versions) do, in fact, incorporate one or more of the "structured" features and other improvements included in COMAL, most commonly the extended form of IF statement.

Thus COMAL should be looked on as taking further and improving on an evolutionary trend which is already occurring in BASIC. Nevertheless, it is fair to say that where BASIC has been extended in the past, it has often been done badly. The wisdom of calling such extended versions by the name of the original language is also questionable.

SUMMARY OF LANGUAGE FACILITIES

The most immediately noticeable feature of a COMAL program is the use of long identifiers. A length of from one up to (at least) 16 letters or digits, beginning with a letter, is permitted. Comparing

```
    10 INPUT H,W,L,T,V (BASIC)
with 10 INPUT HEIGHT, WEIGHT, LENGTH, TIME, VOLUME (COMAL)
```

for example, it is evident that the latter form is much clearer and in itself a considerable aid to making programs "self-documenting". Identifiers in COMAL may contain lower case letters, which are considered equivalent to their upper case equivalents in all respects.

Most of the major BASIC statements are retained in COMAL, in some cases with slight extensions or changes, namely INPUT, PRINT, PRINT USING, IF, FOR ... NEXT, READ, DATA, RESTORE, STOP, END and assignment. The GOTO statement is also retained although it is expected that it will only seldom be necessary to use it.

In addition, multi-line "structured statements", similar to those in Pascal, are provided to facilitate structured programming*. These are of two kinds:

(i) Selection structures

an enhanced form of IF statement

a CASE ... ENDCASE statement (as a more powerful form of the BASIC ON ... GOTO statement)

(ii) Repetition structures

WHILE ... ENDWHILE loops, where a terminating condition is tested at the start of a sequence

REPEAT ... UNTIL loops, where a terminating condition is tested at the end of a sequence.

The FOR ... NEXT statement (retained from BASIC) is also an example of a repetition structure. Provided that GOTO statements are not used, each structured statement has exactly one entry point and one exit point. As a further aid to well-structured programming, named procedures with parameters are provided to replace the use of the BASIC statements GOSUB and RETURN. The string handling facilities of BASIC are also considerably improved in COMAL, which provides arrays of strings as well as the use of substrings, plus a limited form of pattern matching.

The major features of COMAL are illustrated below, in comparison with their BASIC equivalents. A number of minor changes from BASIC will also be noted as they arise.

SELECTION STRUCTURES IN COMAL

(1) *"IF" Statements*

The highly limited "IF" statement of BASIC is replaced by a more general form of single line "IF" statement and a multiline IF ... THEN ... ELSE structure.

Example 1

BASIC

```
100 IF V<=M THEN 140
120 LET M=V
140 .......
```

(Note that in this example it is necessary to test the *negation* of the "condition of interest", i.e. V > M.)

COMAL 100 IF VALUE>MAXVALUE THEN MAXVALUE:=VALUE

Note the form of the assignment statement in COMAL. The keyword LET is not permitted and := replaces the BASIC = sign (to avoid confusion with the use of = as a relational operator). This is probably the point of greatest difference between COMAL and BASIC. In practice, it is likely that many implementations will also permit the BASIC form of assignment, for reasons of compatibility.

The statement following THEN in the above example can be any "Simple Statement" i.e. assignment, GOTO, STOP, END, procedure call or input/output statement.

* Some errors in multi-line statements (e.g. a missing component) cannot, of course, be detected on a line-by-line syntax checking basis. However, these errors are detected when the RUN command is typed, in the same way as "unmatched FOR and NEXT" in BASIC.

Example 2

BASIC

```
100 IF I>100 THEN 160
120 PRINT "O.K."
140 GOTO 200
160 PRINT "TOO HIGH-RESET TO 10%"
180 LET I=10
200 .......
```

COMAL

```
100 IF INTEREST>100 THEN
110    PRINT "TOO HIGH-RESET TO 10%"
120    INTEREST:=10
130 ELSE
140    PRINT "O.K."
150 ENDIF
```

If the condition in line 100 is satisfied, lines 110 and 120 are executed, otherwise line 140 is executed. The statements in both "THEN" and "ELSE" portions of this form of the IF statement can themselves be structured statements, to any depth of "nesting". The ELSE component can be omitted if desired.

Example 3

BASIC

```
 10 REM "NESTING" IF'S CAUSES CONFUSION IN BASIC
 20 IF I>=100 THEN 90
 30 IF I<0 THEN 60
 40 PRINT "THAT WAS O.K."
 50 GOTO 100
 60 PRINT "INTEREST CANNOT BE LESS THAN ZERO!"
 70 GOTO 100
 80 REM HERE THE INTEREST IS >=100
 90 PRINT "INTEREST CANNOT BE SO LARGE"
100 .......
```

COMAL

```
 10 // 'IF' STATEMENTS CAN BE NESTED TO ANY DEPTH
 20 IF INTEREST<100 THEN
 30   IF INTEREST<0 THEN
 40     PRINT "INTEREST CANNOT BE LESS THAN ZERO!"
 50   ELSE
 60     PRINT "THAT WAS O.K."
 70   ENDIF
 80 ELSE
 85   // HERE THE INTEREST IS >=100
 90   PRINT "INTEREST CANNOT BE SO LARGE"
 95 ENDIF
100 .......
```

Note the form of comments in COMAL (lines 10 and 85).

(2) *CASE . . . ENDCASE Statements*

COMAL provides a CASE . . . ENDCASE structured statement as a means of facilitating multiple tests on the value of the same variable.

Example 4

BASIC

```
10 DIM A$(10)
20 PRINT "WHAT NEXT? (BEGIN, END OR CONTINUE)"
```

```
 30 INPUT A$
 40 IF A$< > "BEGIN" THEN 80
 50 PRINT "O.K., LET'S DO IT AGAIN"
 60 REM START AGAIN (STATEMENTS OMITTED)
 70 GOTO 160
 80 IF A$< > "END" THEN 110
 90 PRINT "END OF THIS OPERATION"
100 STOP
110 IF A$< > "CONTINUE" THEN 140
120 PRINT "WE'LL CONTINUE NOW"
130 REM CONTINUE HERE (STATEMENTS OMITTED)
135 GOTO 160
140 REM INPUT WAS NOT 'BEGIN'. 'END' OR 'CONTINUE'
150 PRINT "INVALID CHOICE"
160 REM EXECUTION CONTINUES HERE
```

COMAL

```
 10 DIM ANSWER$ OF 10
 20 PRINT "WHAT NEXT? (BEGIN, END OR CONTINUE)"
 30 INPUT ANSWER$
 40 CASE ANSWER$ OF
 50 WHEN "BEGIN"
 60   PRINT "O.K., LET'S DO IT AGAIN"
 70   // START AGAIN (STATEMENTS OMITTED)
 80 WHEN "END"
 90   PRINT "END OF THIS OPERATION"
100   STOP
110 WHEN "CONTINUE"
120   PRINT "WE'LL CONTINUE NOW"
130   // CONTINUE HERE (STATEMENTS OMITTED)
140 OTHERWISE
150   // INPUT WAS NOT 'BEGIN', 'END', OR 'CONTINUE'
160   PRINT "INVALID CHOICE"
170 ENDCASE
180 // EXECUTION CONTINUES HERE
```

(Note the form of the string dimension statement in BASIC. This will be discussed further later.) The value of ANSWER$ is examined. If it is the string "BEGIN", the statements between 50 and 80 are executed. If it is "END", statements 90 and 100 are executed, and so on. The final component in the structured statement, beginning OTHERWISE, is executed if ANSWER$ is none of the values "BEGIN", "END" or "CONTINUE".

Each component (WHEN or OTHERWISE) can comprise any number of statements, including structured statements. The OTHERWISE component can be omitted.

The keyword WHEN can also be followed by a list of possible values, as shown in the example below.

Example 5

COMAL

```
10 DIM LETTER$ OF 1
20 INPUT LETTER$
30 // READ A SINGLE CHARACTER
40 CASE LETTER$ OF
50 WHEN "A", "E", "I", "O", "U"
60   PRINT "CHARACTER IS A VOWEL"
70 WHEN "X", "Y", "Z"
80   PRINT "CHARACTER IS X, Y OR Z"
90 ENDCASE
```

The value of a numeric variable can also be tested, as in Example 6.

Example 6

```
10 // READ IN AN INTEGER LESS THAN 100
20 INPUT NUMBER
30 CASE NUMBER OF
40 WHEN 1,4,9,16,25,36,49,64,81
50   PRINT "A SQUARE"
60 OTHERWISE
70   PRINT "NOT A SQUARE"
80 ENDCASE
```

REPETITION STRUCTURES IN COMAL

(i) The FOR ... NEXT structure is retained from BASIC, but with a slightly different syntax, e.g.

```
10 FOR INDEX:=5 TO 99 STEP 2 DO
   .......
50 NEXT INDEX
```

A single-line FOR statement is also available, for example

```
10 DIM COUNT(50)
20 FOR INDEX:=1 TO 50 DO COUNT(INDEX):=0
```

In this form of the statement, the keyword "DO" can be followed by any "simple statement".

(ii) The WHILE ... ENDWHILE structured statement provides an alternative form of loop structure where execution continues until a specified condition is satisfied at the beginning of the loop.

Example 7

BASIC

```
 10 REM CALCULATE SQUAREROOTS
 20 LET X=10
 30 LET D=X
 40 LET S=X/2
 50 REM ITERATION
 60 IF ABS(D)<=0.001 THEN 100
 70 LET D=(X/S-S)/2
 80 LET S=S+D
 90 GOTO 60
100 PRINT S
```

COMAL

```
10 // CALCULATE SQUAREROOTS
20 X:=10; DELTA:=X; SQRT:=X/2
30 // ITERATION
40 WHILE ABS(DELTA)>0.001 DO
50   DELTA:=(X/SQRT-SQRT)/2
60   SQRT:=SQRT+DELTA
70 ENDWHILE//END OF ITERATIONS LOOP
80 PRINT SQRT
```

Statements 50 and 60 are executed repeatedly provided that the condition in line 40 is satisfied. Thus the test for performing the loop is carried out at the beginning. Note that multiple assignments are permitted in a single line (separated by semicolons). Also note the comment on line 70 following ENDWHILE. A single line form of the WHILE statement is also available, e.g.

```
10 // FIND 'UNITS' PART (0-9) OF A GIVEN INTEGER
20 WHILE NUMBER>9 DO NUMBER:=NUMBER-10
```

The keyword DO can be followed by any "simple statement" in this form of the statement.

(iii) The REPEAT ... UNTIL structure is similar to WHILE ... ENDWHILE, but in this case the test for termination of the loop is made at the end, with the loop only continuing if the given statement is NOT satisfied. Thus the loop must always be performed at least once. No single line form of the statement is available.

Example 8

BASIC

```
10 DIM A$(1)
20 PRINT "SHALL WE CONTINUE? ANSWER Y OR N";
30 INPUT A$
40 IF A$="Y" OR A$="N" THEN 60
50 GOTO 20
60 .......
```

COMAL

```
10 DIM ANSWER$ OF 1
20 REPEAT
30   PRINT "SHALL WE CONTINUE? ANSWER Y OR N";
40   INPUT ANSWER$
50 UNTIL ANSWER$="Y" OR ANSWER$="N"
60 .......
```

Note that COMAL permits lines 30 and 40 to be combined into a single statement e.g.

```
30 INPUT "SHALL WE CONTINUE? ANSWER Y OR N": ANSWER$
```

with the given string then used as a "prompt" for input.

STRINGS IN COMAL

COMAL allows arrays of strings to be declared, e.g.

```
10 DIM NAME$(5) OF 10
```

declares an array of 5 elements, each a string of up to 10 characters in length. The notation NAME$(4) refers to the fourth (complete) string in the array. "Simple strings" of characters can also be declared, e.g.

```
20 DIM ONENAME$ OF 10
```

Both the number of elements in a string array and the maximum string length ("dimension") can be specified as the value of arithmetic expressions. The following notation is used to select a substring in each case

String array NAME$(4,3:5)
Simple string ONENAME$(3:5)

In each case the substring selected begins at the third character and is of length five characters. The notation NAME$(4,3) denotes the substring of NAME$(4) starting at the third character and of length one.

Example 9

```
10 DIM NAME$(5) OF 10, INNAME$ OF 10
20 INDEX:=1
30 INPUT INNAME$
40 WHILE INNAME$<>"STOP" AND INDEX<=5 DO
50   NAME$(INDEX):=INNAME$;INDEX:=INDEX+1
60   INPUT INNAME$
70 ENDWHILE
```

The program reads in up to five names to string array NAME$. Input stops after the fifth name or when STOP is input.

Two further useful facilities are provided in COMAL:

(i) A string "concatenation" operator.
Thus if FIRST$ is "ALPHA" and SECOND$ is "BETA" then

```
JOINT$: = FIRST$ + SECOND$
```

gives JOINT$ as "ALPHABETA".

(ii) A simple form of pattern matching. For example in the sequence

```
10 DIM VOWEL$ OF 5, CHAR$ OF 1
20 VOWEL$: = "AEIOU"
30 INPUT CHAR$
40 IF CHAR$ IN VOWEL$ THEN PRINT "CHARACTER IS A VOWEL"
```

The condition in line 40 is satisfied if CHAR$ is contained anywhere in VOWEL$. In general, IF FIRST$ IN SECOND$ THEN . . . is satisfied if the characters of FIRST$ form a consecutive substring of SECOND$.

PROCEDURES IN COMAL

Named procedures are provided in COMAL as a superior replacement for the "GOSUB" subroutines of BASIC. For example, the calculation of squareroots given previously can be converted into a subroutine (BASIC) or a procedure (COMAL) as follows.

Example 10

BASIC

```
1010 REM CALCULATE SQUARE ROOT OF X
1020 LET D=X
1030 LET S=X/2
1040 REM ITERATION
1050 IF ABS(D)< =0.001 THEN 1090
1060 LET D=(X/S-S)/2
1070 LET S=S+D
1080 GOTO 1050
1090 RETURN
```

COMAL

```
1010 PROC CALCROOT(X, REF SQRT)
1020 DELTA: =X:SQRT: =X/2
1030 // ITERATION
1040 WHILE ABS(DELTA)>0.001 DO
1050   DELTA: =(X/SQRT-SQRT)/2
1060   SQRT: =SQRT+DELTA
1070 ENDWHILE
1080 ENDPROC CALCROOT
```

The procedure can be "invoked" to calculate the square root of 10, say, and return it as the value of the variable ROOT10 by means of the statement

```
10 EXEC CALCROOT(10,ROOT10)
```

There are two principal advantages of the COMAL procedure over the BASIC form of subroutine.

(i) Like all other constructs in COMAL, a procedure has only one entry point and one exit point. By contrast, there is nothing to prevent an incorrect entry into a BASIC subroutine, e.g. by GOSUB 1030 or by "falling through" from line 1000 say.

(ii) COMAL permits parameters to be passed to a procedure (X and SQRT in the above example). Thus to find the squareroot of a different variable, say Y, and to return its value in variable Z say, it is only necessary to invoke the procedure by

```
EXEC CALCROOT(Y,Z).
```

An equivalent set of statements in BASIC would be

```
10 LET X=Y
20 GOSUB 1010
30 LET Z=S
```

The word REF in line 1010 of the COMAL version indicates that parameter SQRT is passed "by reference". Parameters not preceded by REF are passed "by value". Thus when CALCROOT is invoked by

```
10 EXEC CALCROOT(10,ROOT10)
```

the value 10 is used for X in lines 1020 and 1050 of the procedure, whereas the assignments to parameter SQRT in lines 1020 and 1060 affect the value of the corresponding variable (ROOT10) in the main program. Numeric and string variables, arrays and expressions are permitted as parameters. Procedures can also call themselves recursively.

A function can be defined as a special form of procedure for use in an arithmetic expression, such as 10 TOTAL: =TOTAL+SUMSQ(A,B,C). In this case the procedure definition must contain at least one assignment statement with the procedure name on the left-hand side. Standard functions such as SIN, COS, TAN, ABS, EXP etc. are also available. The most significant difference between COMAL procedures and those in languages such as FORTRAN and ALGOL-68 is that all variables in a procedure (excluding parameters) are "global", i.e. variable DELTA in the above example is identical to any other use of DELTA in the remainder of the program. If DELTA existed and had a value before CALCROOT was executed, its value would be changed by the execution of the procedure (and remain changed afterwards). This global property of variables can be a handicap, since the user of a procedure may need to know which variables are changed when it is executed. On the other hand, it is easy to use such procedures to "structure" a program to aid readability, e.g.

```
10 IF VALUE>0 THEN
20   EXEC THISPATH
30 ELSE
40   EXEC THATPATH
50 ENDIF
```

without having to pass every variable used in "THISPATH" or "THATPATH" as a parameter.

It may be expected that many implementors of COMAL 80 will add a facility to declare a procedure as "closed", in which case all variables (apart from parameters) are purely 'local' to the procedure in which they are used.

TWO COMPLETE COMAL PROGRAMS

To complete this description of the COMAL language here are two longer examples of COMAL programs, which are intended to be self-explanatory, the first non-numeric and the second numeric.

Example 11

```
10 // INPUT STRINGS, FINISHING WITH "END", STORE IN A STRING
   ARRAY
20 // AND PRINT OUT WITH ALL DIGITS 0-9 REPLACED BY ASTERISKS
30 INPUT "HOW MANY WORDS AT MOST?":N
40 INPUT "HOW LONG IS EACH ONE(MAXIMUM)?": L
50 DIM WORD$(N) OF L, ONEWORD$ OF L,DIGITS$ OF 10
60 DIGITS$= "1234567890"
70 EXEC TAKEIN
80 FOR I:=1 TO MAX DO EXEC REPLACE(WORD$(I))
90 FOR I:=1 TO MAX DO PRINT WORD$(I)
100 END
110 PROC TAKEIN
120   I:=0
130   INPUT "FIRST WORD":ONEWORD$
140   WHILE ONEWORD$< >"END" DO
150     I:=I+1;WORD$(I):=ONEWORD$
160     INPUT "NEXT WORD":ONEWORD$
170   ENDWHILE
```

```
180   MAX:=I
190 ENDPROC TAKEIN
200 PROC REPLACE(REF THISWORD$)
210   NCHARS:=LEN(THISWORD$)
220   FOR INDEX:=1 TO NCHARS DO
230     IF THISWORD$(INDEX) IN DIGITS$ THEN THISWORD$(INDEX):="*"
240   NEXT INDEX
250 ENDPROC REPLACE
```

Example 12

```
 10 // PROGRAM TO SOLVE A SEQUENCE OF QUADRATIC EQUATIONS '
 20 // A*X↑2+B*X+C
 30 // TERMINATE SEQUENCE BY INPUTTING ZERO VALUE FOR A
 40 INPUT A
 50 WHILE A<>0 DO
 60   INPUT B,C
 70   EXEC FINDROOTS(A,B,C)
 80   INPUT A
 90 ENDWHILE
100 PROC FINDROOTS(A,B,C)
110   // PRINTS ROOTS OF QUADRATIC EQUATION A*X↑2+B*X+C=0
120   SOLUTION DEPENDS ON 'SIGN' OF DISCRIMINANT B↑2-4*A*C
130   DISCRIM:=B↑2-4*A*C
140   SIGNVALUE:=SGN(DISCRIM)
150   CASE SIGNVALUE OF
160   WHEN -1
170     REAL:= -B/(2*A)
180     IMAG:=SQR(-DISCRIM)/(2*A)
190     PRINT "COMPLEX ROOTS";REAL;"+/- i";IMAG
200   WHEN 1
210     X1:=(-B+SQR(DISCRIM))/(2*A)
220     X2:=(-B-SQR(DISCRIM))/(2*A)
230     PRINT "REAL DISTINCT ROOTS";X1;X2
240   WHEN 0
250     X:= -B/(2*A)
260     PRINT "COINCIDENT ROOTS";X
270   ENDCASE
280 ENDPROC FINDROOTS
```

CONCLUSIONS

It is hoped that the above examples give a reasonable indication of the features of COMAL. The language builds on the good start made by BASIC as a general-purpose language for use by the non-expert, especially in an interactive mode. The addition of structured statements derived from Pascal and other features such as string arrays and "long" identifiers removes the major causes of criticism of BASIC and provides a practical evolutionary means of bringing the teaching of programming into the age of "structured programming".

Experience in Denmark strongly suggests that COMAL is well-suited as an introductory programming language for use in schools. It is anticipated that a number of microcomputer implementations will shortly be available in Britain.

Acknowledgements—I should like to thank my fellow members of the Programming Languages Working Party of the British Computer Society Schools Committee—and particularly the chairman, Roy Atherton—for many valuable discussions, and Borge Christensen (Tonder Statsseminarium, Denmark) and Jenny Preece (Open University) for their detailed comments on an earlier version of this paper.

REFERENCES

1. Atherton R., Microcomputers, Secondary Education and Teacher Training. Paper presented at the conference *Microelectronics for Education and Training*, University of Reading (1979).

2. Bowles K., *Problem Solving Using Pascal.* Springer-Verlag, New York (1977).
3. Martinsen P. N., The COMAL programming language. A short description. Dansk Data Elektronik ApS (1980).
4. Østerby T. (Ed.), COMAL 80 Programming Language. Proposal for the nucleus of the COMAL 80 language. COMAL working group, Denmark (1980).
5. Strengths and Weaknesses of BASIC. British Computer Society Schools Committee Programming Languages Working Party (1980).