The Programming Language COMAL - Denmark

(A Brief Description)

By B R Christensen

Data Department,
Tonder Statssem - Ostergrade.

The programming language BASIC would appear to be playing an increasingly prominent role in the teaching of elementary computer science. However, in the eyes of many computer specialists, this represents a highly undesirable state of affairs, since the majority of them have emphatically rejected BASIC as a language suitable for teaching beginners. While the language does represent a measure of improvement upon the 'assembler-like' notations used by desk-top calculators, and indeed, may be useful for certain technical purposes, in the opinion of the author it is a language which is totally unsuitable for use in schools. In such establishments training should be aimed at a general understanding of algorithms and an analysis of their structure. In this respect, the lack of transparency of BASIC has been heavily criticised since it does not permit the underlying structure of an algorithm to show through the program topography. Indeed, BASIC programs tend to mask the structure of an algorithm; anything more ambitious than simple calculations and printout of results soon becomes unreadable to anyone other than the author of the program - and even he, after a short time lapse, would need to invest considerable effort in order to 're-understand' his own program. Of course, this latter difficulty is aggravated by the very restrictive rules for constructing variable names which, in most cases, make it impossible to select names that adequately reflect their meaning.

Based upon a considerable amount of experience, it is the author's view that BASIC programs are 'patchworks' of program fragments knitted together with GOTO statements. Such programs encourage the student to develop bad programming habits. Further, the student is given a superficial picture of an algorithm since he is not motivated into making a careful enough analysis of the structures with which he is working.

After some consideration and many discussions with both colleagues and other workers in the field, the present version of the programming language COMAL (<u>COM</u>mon <u>A</u>lgorithmic <u>L</u>anguage) was designed. The language should be user-orientated and should enable the communication of relatively small algorithms to both man and machine in an understandable manner. At the same time the language should be capable of being implemented as a line-by-line interpreter run on a stand-alone minicomputer. BASIC is used as a host language for COMAL; this may seem somewhat inconsistent after the criticisms made above. However, some of the best BASIC interpreters provide floating-point facilities and some limited operating system capabilities (for example, multiplexing and file handling) both of which are maintained by the manufacturer. Consequently, any implementation of COMAL that could take advantage of such primitives would be both quicker and less costly. While in some ways COMAL is similar to BASIC, it is substantially better. The more important improvements have been inspired by the language PASCAL, designed by N. Wirth (cf. N. Wirth, Systematisches Programmieren, Teubner, 1972). The final version has also been greatly influenced by Benedict Løfstedt, DIAMI, University of Aarhus, with whom the author has had much discussion and whose never failing watchfulness has been a constant source of improvements in the preliminary drafts of the language.

<u>Description of COMAL</u>

Essentially, COMAL is defined as BASIC with the following seven modifications:-

(1) <u>Simple Variables</u>

A simple variable is named according to the rule

$$\boxed{a_1 a_2 a_3 \ldots\ldots a_n \qquad n \leq 5}$$

such that a_1 is a letter while $a_2 \ldots a_n$ represent letters or digits. Two variable names are interpreted as different if they differ in one or more of the five possible characters, for example,

PRICE, VOL, MAX25, MAXI, H235, H 235

(2) <u>Array Variables</u>

Arrays are used in exactly the same way as in BASIC; naming conventions, however, follow the same rule as COMAL simple variables, for example,

STUDT(YEAR,NUM), SUM(TEST(I)), A(2,3)

(3) <u>Control Structures</u>

The *FOR....NEXT* construction is available in COMAL. However, the following constructions, while available in BASIC, are not permitted in COMAL:-

IF P THEN GOTO <u>linenumber</u>

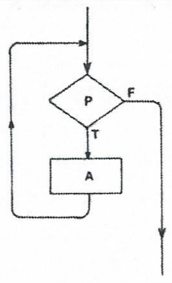
GOTO <u>linenumber</u>

ON I GOTO <u>linenumber</u>

Some of the more powerful control structures available in COMAL are the *WHILE....DO*, *REPEAT....UNTIL*, *IF....THEN*, and *CASE* statement.

WHILE....DO

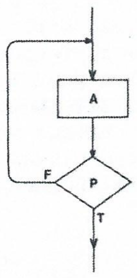
```
WHILE P DO
  --------
  --------
  --------
END WHILE
```



p is a Boolean expression; the program section A is repeated as long as *p* has the value "true". Between *WHILE P DO* and *END WHILE* the program text is indented on the listing (cf. *FOR.... NEXT*).

REPEAT....UNTIL

```
REPEAT
  --------
  --------
  --------
UNTIL P
```



p is a Boolean expression; the program section A is repeated until *p* has the value "true". Between the *REPEAT* and *UNTIL P* the program text is indented on the listing.

There are, then, in effect, only three loop structures in COMAL,

FOR .... NEXT

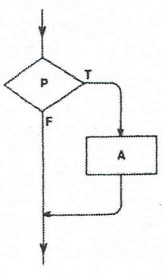
WHILE P DO .... END

and *REPEAT .... UNTIL*

IF....THEN

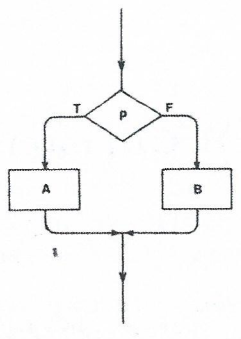
Branching may be achieved by using either of the variants of the *IF....THEN....ELSE* construction. The simplest case is as follows:-

```
IF P THEN DO
  --------
  --------
  --------
END IF
```



The program section A contained between the two control statements is executed only if the Boolean expression has the value "true". After execution of this section, control passes to the statement following the *END IF*. If *p* has the value "false", A is ignored and execution continues with the statement following the *END IF*. Between *IF P THEN DO* and *END IF* the program text is indented on the program listing.

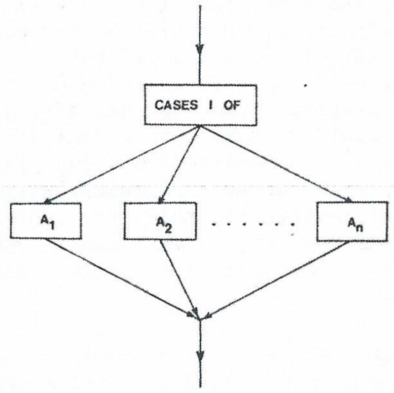
```
IF P THEN DO
  --------
  --------
  --------
ELSE DO
  --------
  --------
  --------
END IF
```



p is a Boolean expression; the program section A between *IF P THEN DO* and *ELSE DO* is executed if *p* has the value "true", otherwise the program section B between *ELSE DO* and *END IF* is executed. After the execution of either section A or section B, execution continues with the statement following the *END IF*. The program text lying between the control statements is indented on the program listing.

CASE statement

```
CASES I OF
CASE a1,a2,...,an
  --------
  --------
  --------
CASE b1,b2,...,bn
  --------
  --------
  --------
...
END CASES
```



I represents an integer expression and $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m, \ldots$ are mutually different integers. The keyword *CASE* may be followed by as many integers as the line-width will permit. The mode of execution of the *CASE* statement is as follows: when the value of I has been determined the list of integer constants following a given *CASE* is examined; if the calculated value is found in the list then the program section A_i, contained between the selected *CASE* and the following *CASE* (or between the last *CASE* and *END CASES* if the case referred to is the last in the block), is executed. If the calculated value of I does not select any of the *CASE*'s the interpreter will set an error flag at runtime.

Boolean Expressions

A Boolean variable is named according to the following rules:-

$$\boxed{a_1 a_2 \ldots a_n\# \qquad n \le 5}$$

where the rules for the string $a_1 a_2 \ldots a_n$ are the same as those for simple variables (see (1) above).

A Boolean variable may be assigned the value "true" or "false". In COMAL, Boolean expressions may be inter-connected by the Boolean operators *AND*, *OR* and *NOT*. The following example illustrates the use of Boolean variables:-

```
LET TEST#= TRUE
LET REMBR#= (MAX=1Ø) AND TEST#
IF REMBR# OR SUM>Ø THEN DO
```

(5) Statements containing more than one Assignment

LET-statements in COMAL are allowed to contain more than one assignment, for example,

```
LET SUM=Ø, A=1, B=A/2+SUM, TEST#=TRUE
```

The above multiple assignment is equivalent to four separate *LET*-statements. A *LET*-statement may contain as many assigenements as the line-width permits.

Note: In some versions of BASIC the keyword *LET* may be omitted from the beginning of the assignment (for example, Data General's EXTENDED BASIC).

(6) Facilities for editing Programs

The interpreter automatically supplies lin-numbers for the statements as they are typed in. Unless the programmer specifies otherwise, the default numbering sequence is 10, 20, 30, ... (as is the practice with many versions of BASIC, for example, that available on HP 9030 machines). Editing follows the same conventions as are used in BASIC. In order to produce a listing of a program, two *LIST*-commands are available: *$LIST* and *$LISTNUM*. A $ character in the first character position of a statement is interpreted as signifying an immediate command; such a command is not stacked, but instead, is immediately executed. *$LIST* may be used to produce a program listing without line-numbers while *$LISTNUM* causes line-numbers to be printed with the listing. Since there are no *GOTO*-statements in COMAL, line numbers are only required for editing purposes.

(7) Subroutines

The statement

GOSUB linenumber

is not permitted in COMAL. Instead, a subroutine may be activated by the statement,

$$\boxed{\mathit{CALL}\ \underline{name}\ (\underline{arg_1}, \underline{arg_2}, \ldots, \underline{arg_n})}$$

in which *name* is the name of a subroutine and $arg_1, arg_2, \ldots, arg_n$ is a list of arguments, the values of which are assigned to a corresponding set of parameters in the subroutine.

The definition of a function is initiated by means of the statement,

$$\boxed{\mathit{SUB}\ \underline{name}(\underline{par_1}, \underline{par_2}, \ldots, \underline{par_n})}$$

where *name* is the name of the subroutine and $par_1, par_2, \ldots, par_n$ is a set of parameters used by the subroutine; these parameters are assigned the values of a corresponding set of arguments given in the *CALL*-statement. Apart from this, subroutines are handled as in BASIC.

Illustrative Program Listings

In order that the format and structure of COMAL programs may be illustrated, program listings A through D show some simple algorithms implemented in COMAL.

A

```
INPUT N
D=1
REPEAT
  D=D+1, DIVI#=(N/D=INT(N/D))
UNTIL DIVI# OR D*D>N
IF  DIVI# THEN DO
  PRINT "SMALLEST PRIMEDIV. IS:", D
ELSE DO
  PRINT N, " IS A PRIME NUMBER"
END IF
```

B

```
INPUT LEFT,RIGHT
COOR=Ø, STEP=Ø, HOME=Ø
REPEAT
  NUM=RND(Ø), STEP=STEP+1
  IF NUM<5 THEN DO
       COOR=COOR+1
  ELSE DO
       COOR=COOR-1
  END IF
  IF COOR=Ø THEN DO
       HOME=HOME+1
  END IF
UNTIL COOR=RIGHT OR COOR=LEFT
PRINT COOR, HOME, STEP
```

C

```
INPUT P1,N
QUEUE=Ø, TIME=Ø
FOR I=1 TO N
  IF QUEUE<>Ø THEN DO
    TIME=TIME+1
    IF TIME>3 THEN DO
      QUEUE=QUEUE-1
      IF QUEUE>Ø THEN DO
        TIME=1
      ELSE DO
        TIME=Ø
      END IF
    END IF
  END IF
  NUM=RND(Ø)
  IF NUM<P1 THEN DO
    QUEUE=QUEUE+1
    IF TIME=Ø THEN DO
       TIME=1
    END IF
  END IF
  PRINT I, QUEUE, TIME
NEXT I
```

D

```
REM ********* MAIN PROCEDURE *********
REM VAR: A,B,F ARE RESERVED SUB HORNER(A,B)
PRINT "ENTER DEGREE OF POLYNOMIAL"
INPUT GRAD
DIM A(GRAD), REAL(GRAD), CMPLX(GRAD)
PRINT "ENTER COEFFICIENTS OF POLYNOMIAL"
FOR I=1 TO GRAD
 PRINT USING "A(##)",I
 INPUT A(I)
NEXT I
IF A(Ø)=Ø THEN DO
  PRINT "YOUR POLYNOMIAL IS NOT OF DEGREE ",GRAD
  STOP
END IF
N=Ø
REM
REM ========= START SEARCHING ======================
WHILE GRAD>Ø DO
  IF A(GRAD)=Ø THEN DO
    N=N+1, GRAD=GRAD-1
    REAL(N)=Ø, CMPLX(N)=Ø
  ELSE DO
    XCTR=Ø, YCTR=Ø
    FMIN=A(N)↑2  FCTR=FMIN
    DX=ABS(A(N)/A(Ø))↑(1/N), DY=Ø
    REPEAT
      FOR I=1 TO 4
        U=DY, DY=DX, DX=U
        X=XCTR+DX, Y=YCTR+DY
        CALL HORNER(X,Y)
        IF F<FMIN THEN DO
          XMIN=X, YMIN=Y, FMIN=F
        END IF
      NEXT I
      IF FMIN<FCTR THEN DO
        DX=1.5*DX, DY=1.5DY
        XCTR=XMIN, YCTR=YMIN, FCTR=FMIN
      ELSE DO
        U=4*DX-3*DY, DY=4*DY+3*DX, DX=U
      END IF
      U=ABS(XCTR)+ABS(YCTR)
    UNTIL U+ABS(DX)+ABS(DY)=U OR FCTR=Ø
    CALL HORNER(XCTR,Ø)
    IF F<FCTR THEN DO
      GRAD=GRAD-1, N=N+1
      REAL(N)=XCTR, CMPLX(N)=Ø
      U=Ø
      FOR J=Ø TO GRAD
        U=U*XCTR+A(J), A(J)=U
      NEXT J
    ELSE DO
      GRAD=GRAD-2, N=N+2
      REAL(N)=XCTR, REAL(N-1)=XCTR
      CMPLX(N)=YCTR, CMPLX(N-1)=-YCTR
      U=Ø, V=Ø, K=2*XCTR, M=XCTR↑2+YCTR↑2
      FOR J=Ø TO GRAD
        W=A(J)+K*U-M*V, A(J)=W, V=U, U=W
      NEXT J
    END IF
  END IF
END WHILE
REM ========== SEARCH ENDED ==================================
REM
REM
REM ========== OUTPUT ROOTS ==================================
PRINT "REAL ROOTS"
FOR I=1 TO N
  IF CMPLX(I)=Ø THEN DO
    PRINT USING "-#.####↑↑↑↑", REAL(I)
  END IF
NEXT I
PRINT
PRINT
PRINT "COMPLEX ROOTS"
FOR I=1 TO N
  IF CMPLX(I)<>Ø THEN DO
    PRINT USING "(-#.###↑↑↑↑  , -#.###↑↑↑↑)",REAL(I),CMPLX(I)
  END IF
NEXT I
REM ========== OUTPUT ENDED ==================================
REM
REM
REM ********** SUBROUTINE HORNER **********
SUB HORNER(A,B)
U=Ø, V=Ø, K=2*A,M=A↑2+B↑2
FOR J=1 TO GRAD
  W=A(J)+K*U-M*V, V=U, U=W
NEXT J
F=(A(N)+U*A-M*V)↑2+(U*B)↑2
RETURN
REM ********** END SUBROUTINE HORNER ******
REM
REM
REM ********** END OF PROGRAM *************
```

National Development Programme in Computer Assisted Learning (cont'd)

tween classroom teachers and computer educators, criteria for selecting the content of data bases and for justifying the use of the computer.

In September 1975, 350 children from 5 junior schools in South Glamorgan are to take part in a computer managed remedial reading project. This development project, a joint effort with South Glamorgan County Council, is funded by the National Programme for two and a quarter years from the 1st October 1974 at a cost of £90,366. The aim of this computer managed system is to assist classroom teachers in developing and monitoring individualised courses of study for children experiencing difficulties with reading. In the first year existing teaching materials will be classified and arranged, in-service education for participating teachers implemented and computer procedures set up using the ICL 1900 series computer at City Hall, Cardiff. The educational and the computer models were specified in earlier design studies funded by the National Programme.

A further dozen development projects and feasibility studies lie in the other areas of the Programme with the majority in tertiary education. Projects are located in many parts of the United Kingdom.

Conclusion

In terms of its funding relationship with educational institutions, the National Devlopment Programme is probably the only national programme of its type in the world. In the UK it is also the largest programme ever mounted to develop the use of a single educational technology.

For further details apply, in the first instance, to the International Information Centre - address page 2.

A Neglected Dimension in School Computing (cont'd)

Field F

This field acts as a 'sex-indicator'. This is automatically set to F for females and M for males. If it is found desirable, other letters could equally well be used.

Operational Details

EFCP is designed to be a self-instructional system. However in some cases, as for example when one wishes to modify the name generation section of the system, more detailed information is required by the user. In such cases the user should consult the EFCP User's Manual.