

Børge R. Christensen

C O M A L 80

SYNTAX DIAGRAMS WITH COMMENTS

INTRODUCTION

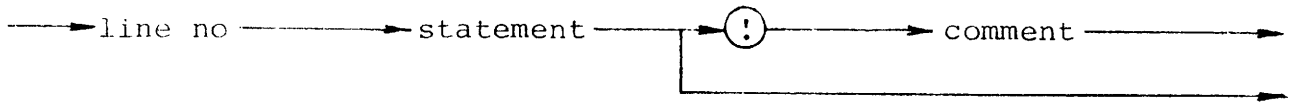
COMAL (COMmon Algorithmic Language) was designed in 1974 by Benedict Løfstedt, Dept. of Computer Science, University of Aarhus, and the author of this report, Børge R. Christensen at DATO, the States Teacher Training College, Tonder, Denmark.
COMAL may be considered as an "additive" to BASIC, to make it possible for the programmer to write well structured and easy readable programs and still have the good features of BASIC at his disposal, i.e. the I/O, interactivity and immediate syntax control.
The first version of COMAL was implemented at DATO in 1975, based on a DATA GENERAL XBASIC by Per Christiansen and Knud Christensen. This version has since then been in general use in Danish Schools.
COMAL 80 includes COMAL 75 and some version of BASIC. If two COMAL 80 versions differ, it will always be on the BASIC part, since the COMAL extensions are well defined. Only structured BASIC which includes these extensions may be called COMAL or "Structured BASIC/COMAL". The COMAL 80 has been carefully designed to meet the needs of users as observed through almost five years of work with COMAL 75.

Tonder, September 1979 Børge R. Christensen.

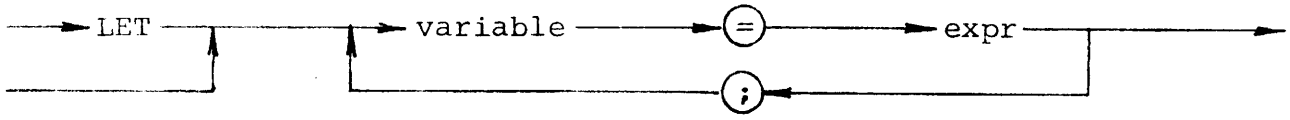
COMAL 80

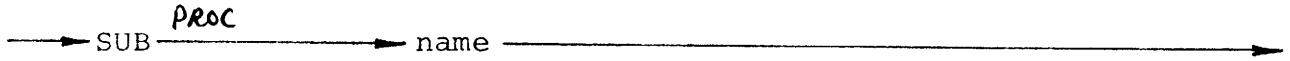
SYNTAX DIAGRAMS

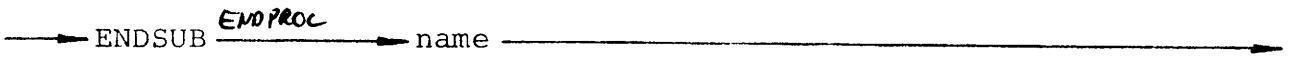
line:

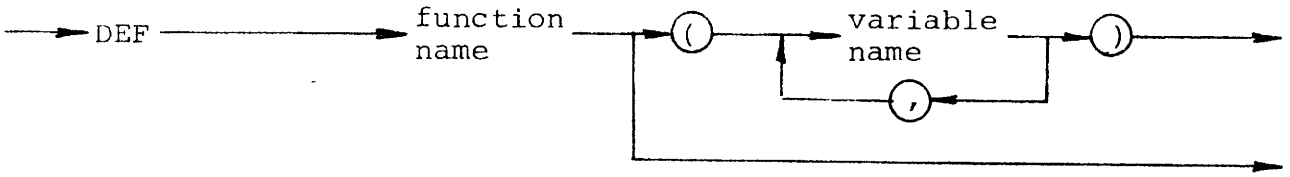


statement:

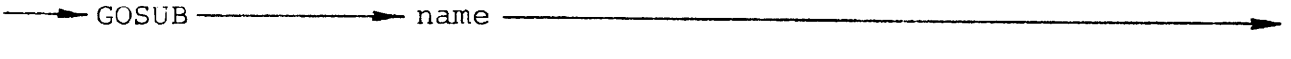


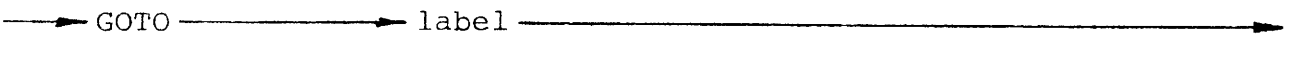


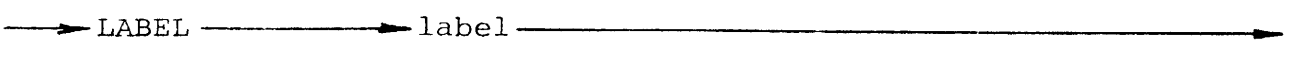


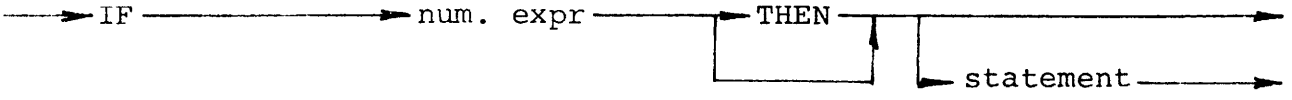


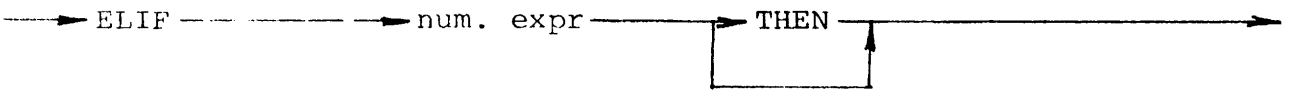


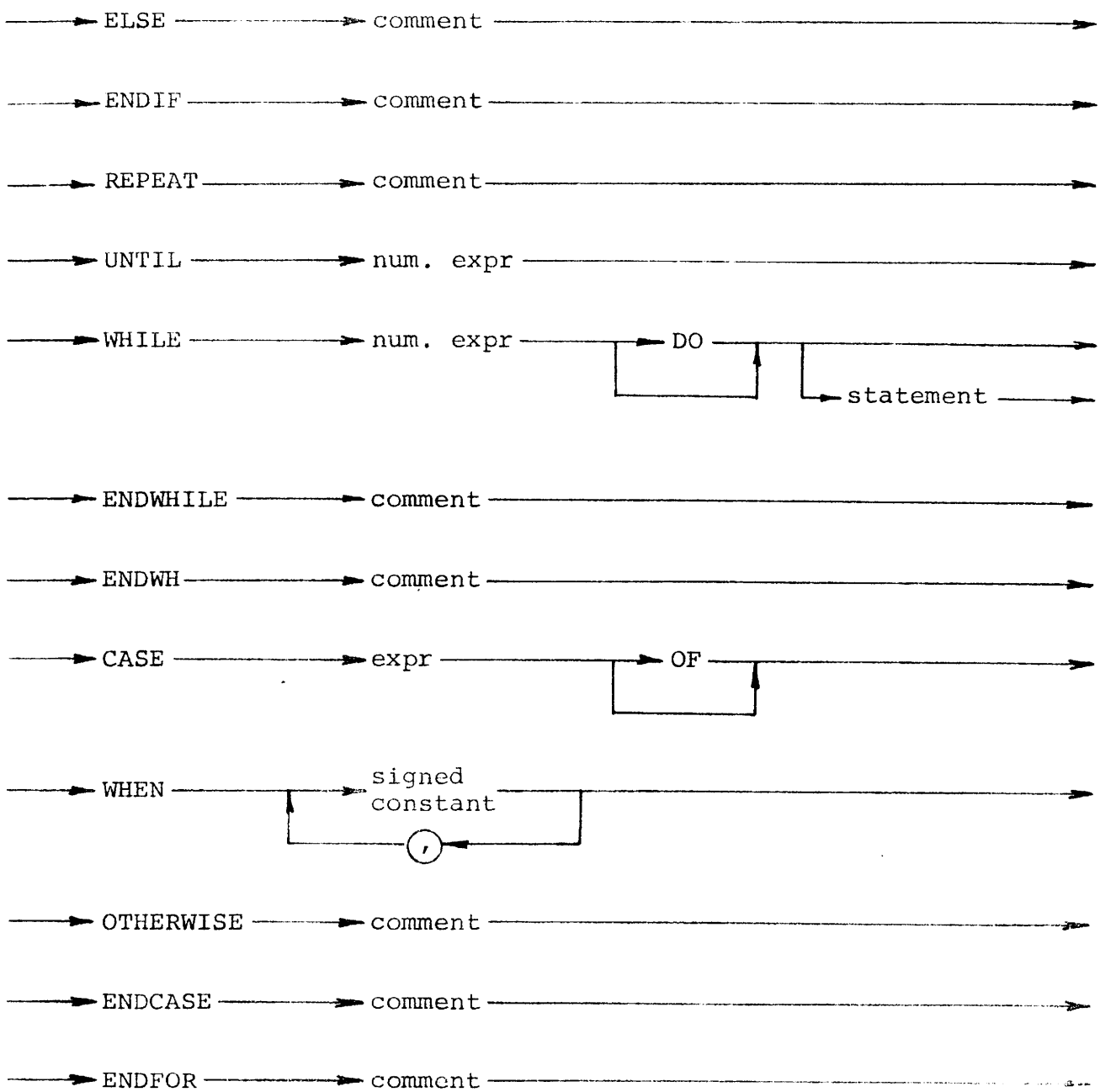




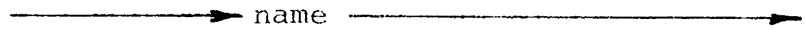




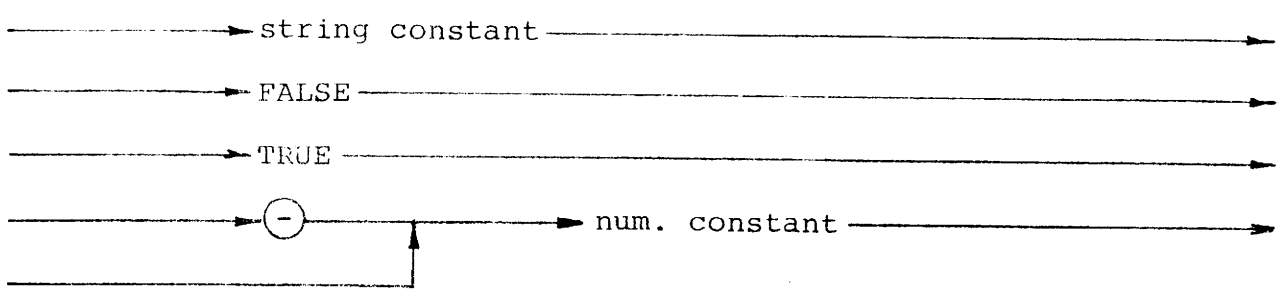



ELSE
comment
ENDIF
comment
REPEAT
comment
UNTIL
num. expr
WHILE
num. expr
DO
statement
ENDWHILE
comment
ENDWH
comment
CASE
expr
OF
WHEN
signed
constant
,
OTHERWISE
comment
ENDCASE
comment
ENDFOR
comment

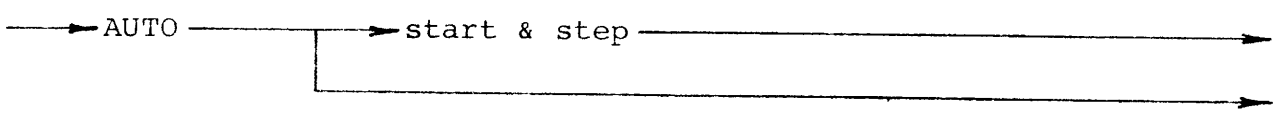
label:


name

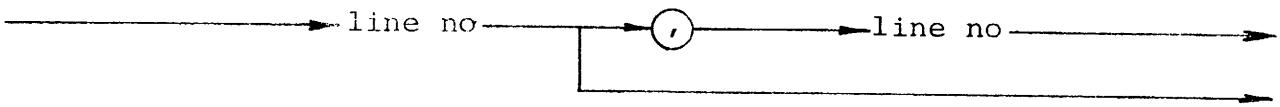
signed constant:



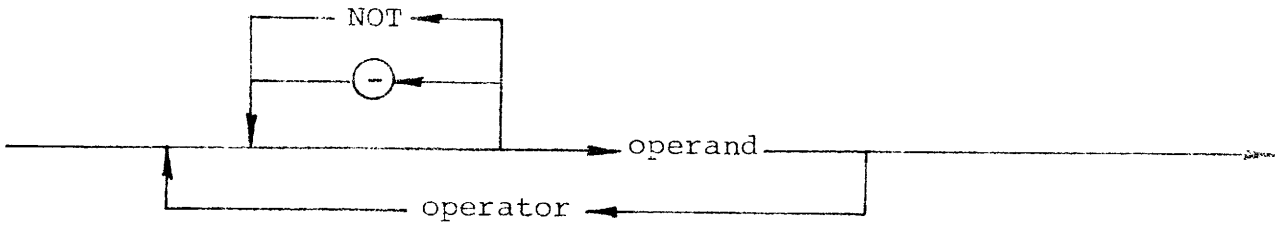
command:



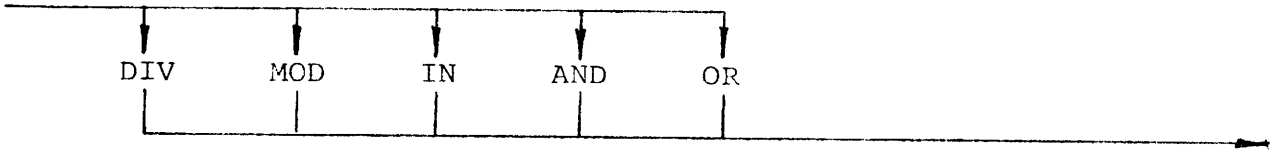
start & step:



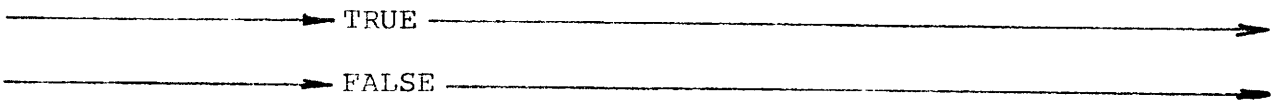
(string, real, (int), bool) expr:

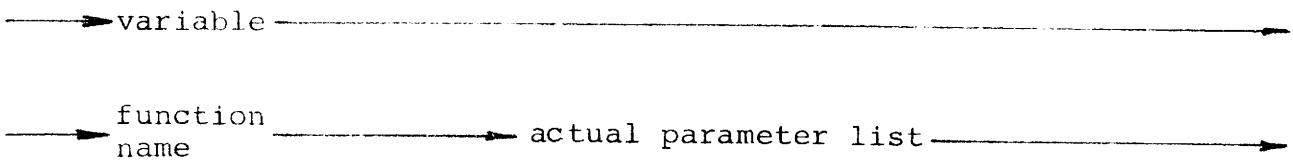


operator:

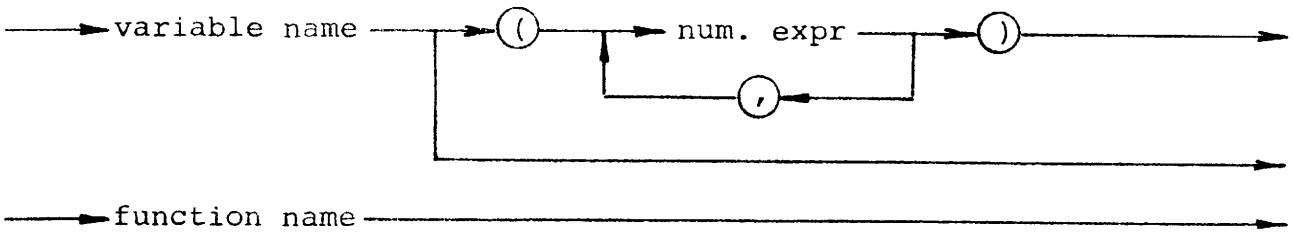


operand:

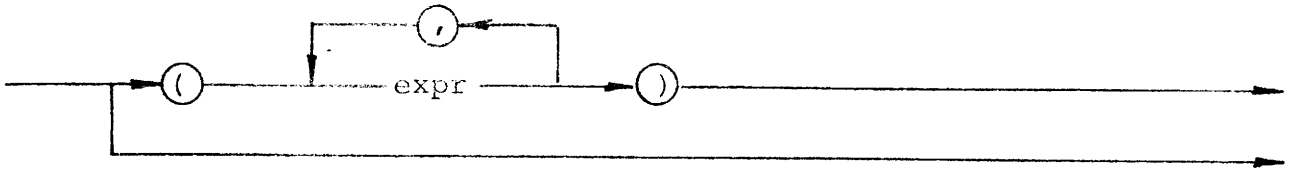




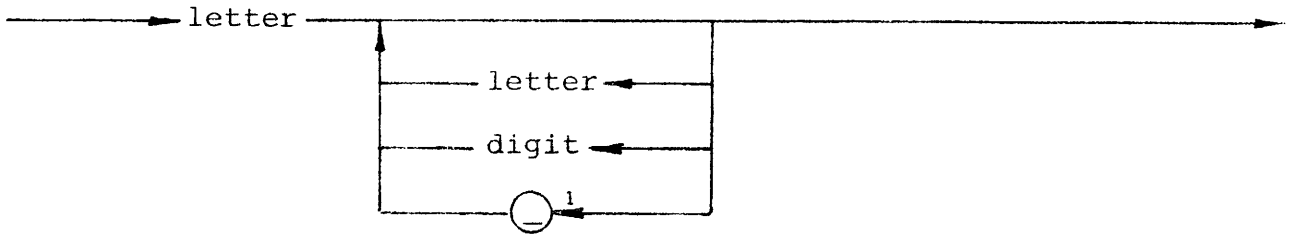
variable:



actual parameter list:

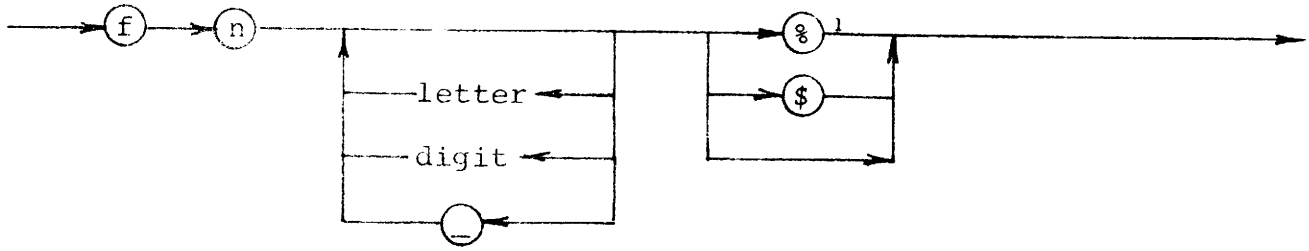


name:



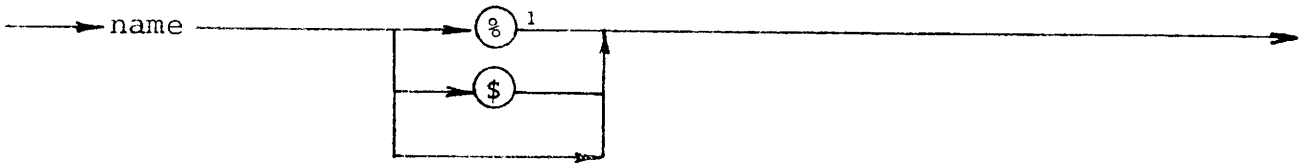
[1]) or some other character like '. Not in all COMAL versions.

function name:



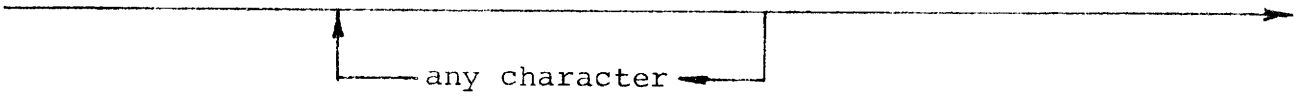
[1]) Type integer (%) and ($) not in all COMAL versions.

variable name:



[1]) Type integer (%) not in all COMAL versions.

comment:



COMMENTS TO

THE SYNTAX DIAGRAMS

INTRODUCTION

The following exposition will deal mainly with the statements that define COMAL. BASIC statements will be mentioned only where they have been subject to extensions due to the definition of COMAL. Since the COMAL statements have been introduced to facilitate structured programming, the syntax diagrams can only unveil very little of the true power of these statements that have to be seen in global contexts. Great care will therefor be taken to display the structures controlled by the COMAL statements.

LET

A LET statement will take as many assignments as the line with permits, each individual assignment being separated from the next one by means of the semicolon (;).

GOSUB, SUB, ENDSUB

If a part of a program is initiated with the statement

```
SUB "name"
```

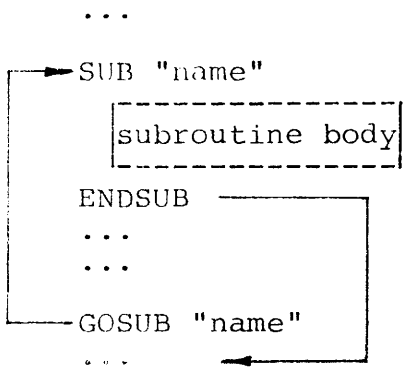
where "name" is a string formatted as defined in the syntax diagrams, and is terminated with the statement

```
ENDSUB
```

this part may be called as a subroutine using the statement

```
GOSUB "name"
```

When the subroutine has been executed, control is passed to the statement following the GOSUB statement that called the subroutine.
The program text between the SUB and ENDSUB statements is indented in the program listing.
If the interpreter reads a SUB statement merely as a result of the sequential execution of the program lines and not as a result of a call, it takes no notice of the subroutine, but goes on with the statement following the matching ENDSUB.



DEF, ENDDEF

If a subprogram is initiated with the DEF-statement and terminated with the ENDDEF statement, it may be used as a function. All variables introduced in the lines between DEF and ENDDEF are local, and gobal variables cannot be accessed from these lines. Parameters may be simple variables of any type, and they are all called by value. The output from the function is returned through the functions name. Thus an assignment like this:

function name:=expression of correct type

must appear somewhere in the body of the function.

Example.

```
DEF FNGCD%(X%,Y%)
  ...
  ...
  LET FNGCD%=A%
ENDDEF FNGCD
```

This function is used in the statement:

```
IF FNGCD%(A%,B%)=1 THEN PRINT "A AND B ARE REL. PRIMES."
```

□□□

GOTO, LABEL

Adresses for GOTO may be given by labels in COMAL. In some versions of COMAL a label may also be used by the RESTORE statement. Thus the statement:

```
RESTORE NAMES_OF_PERSONS
```

will set the data pointer to the first element of the queue

defined by the DATA statements following the statement:

```
LABEL NAMES_OF_PERSONS
```

IF, ELIF, ELSE, ENDIF

Note: In all COMAL versions a numerical expression is in proper context considered false, if it has a value of 0, and true in all other cases.

The four statements provide the following:

a. IF .. ENDIF

```
IF expr THEN
   ┌----------┐
   |    A     |
   └----------┘
ENDIF
```

If the expression has a value equivalent to true, program section A is executed. If the expression evaluates to false, program section A is ignored.
The program text between IF and ENDIF is indented in the program listing (cf. FOR .. NEXT i most BASIC versions).

b. IF .. ELSE .. ENDIF

```
IF expr THEN
   ┌----------┐
   |    A     |
   └----------┘
ELSE
   ┌----------┐
   |    B     |
   └----------┘
ENDIF
```

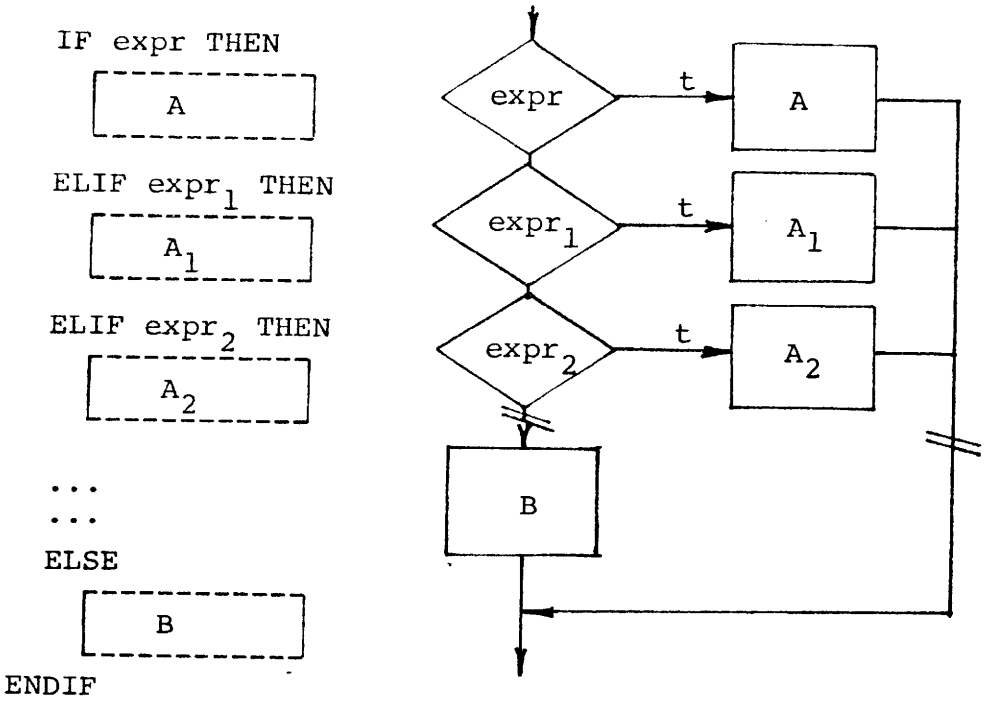
If the expression evaluates to true, program section A is executed. If the expression has a value equivalent to false, program section B is executed.
The program text between the control statements is indented in the program listing.

c. IF .. ELIF .. ELIF .. .. ELSE .. ENDIF

(diagram on next page).

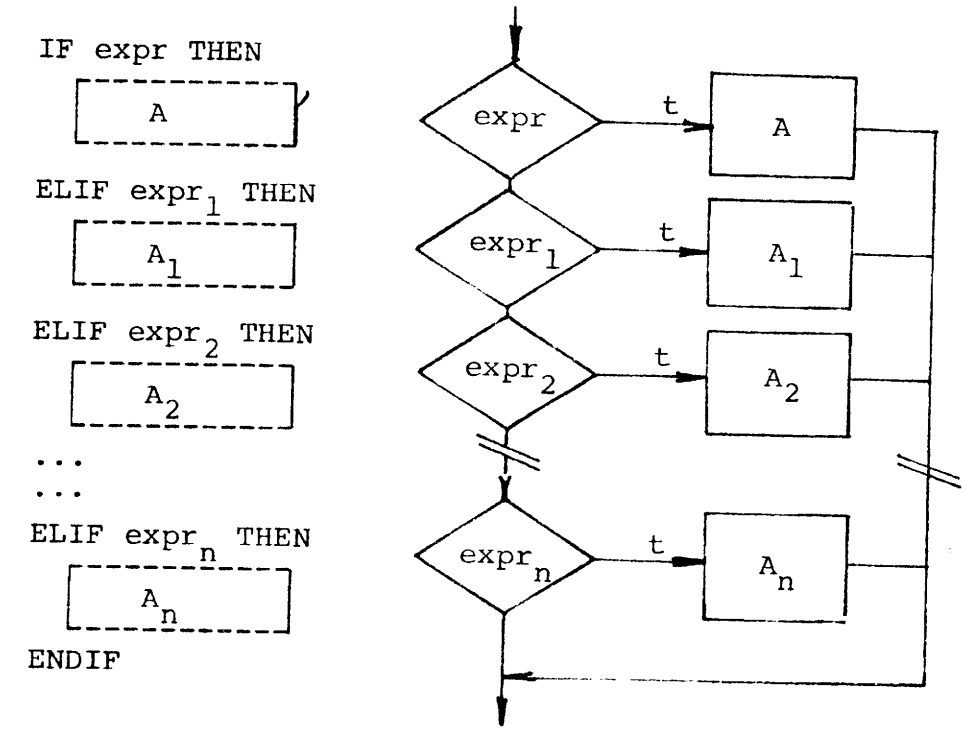
The keyword ELIF is an abbreviation of ELSE IF. As the flowchart on the next page will show, only one of the processes described in the structure is executed. Also

note that if more than one of the expressions may be evaluated to a value of true, only the first one will trigger off a process.



If the final alternative ELSE is left out, you get

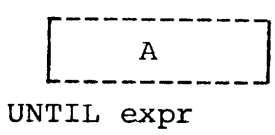
d. IF .. ELIF .. ELIF .. .. ENDIF



As may be seen from the syntax diagrams, the keyword THEN is optional. If it is not entered from the keyboard, the COMAL interpreter should insert it automatically for purposes of readability.

REPEAT, UNTIL

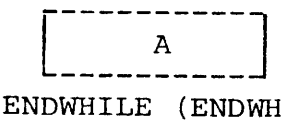
The REPEAT and UNTIL statements provide the following structure:



Program section A is executed repetitively until the expression following UNTIL has a value equivalent to true. When this happens, control passes to the statement following the UNTIL statement.
The program text between REPEAT and UNTIL is indented in the program listing.

WHILE, ENDWHILE, ENDWH

The WHILE and ENDWHILE (ENDWH) statements provide the following structure:



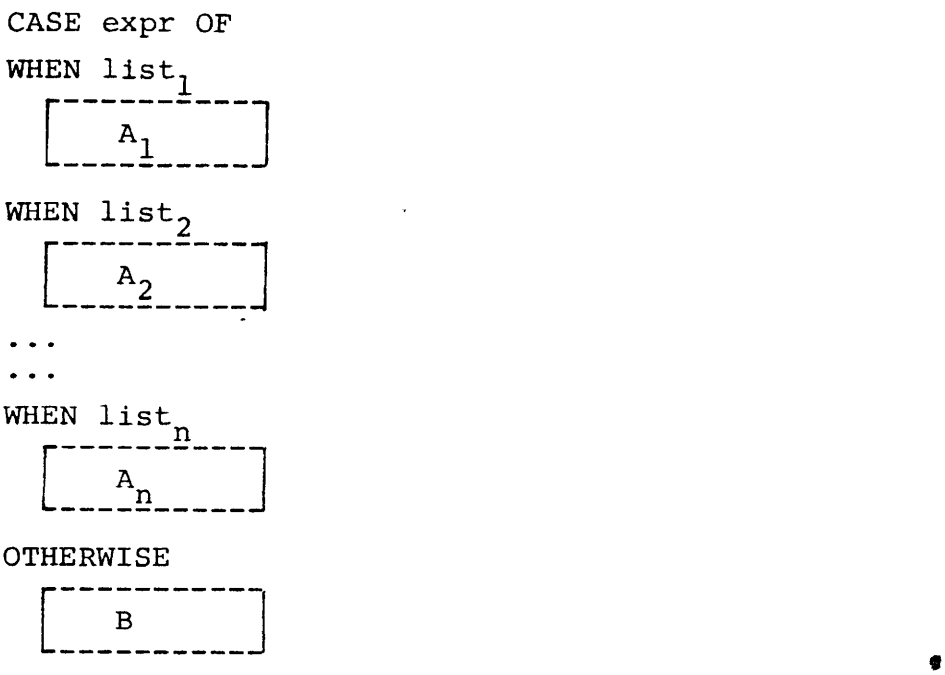
Program section A is executed repetitively while the expression following the WHILE keyword is evaluated to true. When the expression evaluates to false, control passes to the statement following ENDWHILE (ENDWH).
The program text between WHILE and ENDWHILE (ENDWH) statements is indented in the program listing.
The keyword DO is optional, but should be supplied automatically by the interpreter if not entered.

CASE, WHEN, OTHERWISE, ENDCASE

(diagram on next page)

When the expression following CASE has been evaluated,

the list following the first WHEN is examined. If one of the constants in this list is equal to the value of the expression, program section A_1 is executed, and control is then passed to the statement following ENDCASE. If no such item is found, the list following the second WHEN is examined. If the value of the expression is found, A_2 is executed, and control is then passed to the statement following ENDCASE. If the value has still not been found, the interpreter starts on the third list, etc.
A default case (program section B) may be inserted and is executed if the value of the expression is not found in any of the lists following the WHEN keywords. The default case is indicated be the keyword OTHERWISE.



The OTHERWISE case may be left out, but the interpreter will then stop the execution of the program with an error message if no constant corresponding to the value of the expression has been found in the "WHEN lists".
Note that at most one of the cases is executed. If it so happens that a value of the expression may be found in more than one of the lists, only the first one of these lists will trigger off its process.
The program texts for A_1, A_2, ..., A_n, B are indented in the program listing.
The keyword OF is optional, but should be inserted automatically by the interpreters syntax control if not entered from the keyboard by programmer.

FOR, ENDFOR

ENDFOR may be used in stead of NEXT for reasons of compatibility. The counter variable may or may not occur after NEXT or ENDFOR. The interpreter will in any case look upon it as a comment.
In some versions of COMAL you may use a statement like this:

```
FOR I=10 DOWNTO 1
```

The "stepvalue" is then automatically set to -1.

COMMENTS

Comments are allowed after any statement by using the character "!" to separate the statement from the comment. After all "single keyword statements" like REPEAT, ELSE, ENDWHILE, etc., comments may be inserted without using the separator. This should also be possible after the BASIC keywords END, STOP, and RETURN.

TRUE, FALSE

To improve the readability of the program two constants TRUE and FALSE are predefined. TRUE is equivalent to 1, and FALSE is equivalent to 0.

AND, OR, NOT

In COMAL you have full Boolean algebra at your disposal. As mentioned before a numerical expression is in proper context considered false, if it has a value of 0, and true in all other cases. A Boolean expression like

```
NUMBER>MAX_NUMBER OR NOMORE
```

outputs a value of 1, if it is true, and a value of 0, if it is false. Consequently a statement like this:

```
LET FOUND=(NAME$=STUDENT_NAME$(I))
```

will assign a value of 1 to FOUND, if the condition to the right of the assignment is met, and a value of 0, if not. Thereafter FOUND may be used as if it were a Boolean variable. The "pseudo Boolean" values 0 and 1 should be represented as integers, if the BASIC part of the interpreter includes integers.

IN

The expression

```
<string1> IN <string2>
```

will output a value of 1, if <string1> is found as a sub-string of <string2>, and a value of 0, if it is not found.

Example

If the variable VOWELS$ has been assigned the value: "AEIOUY", the statement

```
IF CHAR$ IN VOWELS$ THEN LET ISVOWEL=TRUE
```

will assign a value of 1 ("true") to ISVOWEL, if CHAR$ has the value of a vowel.

□□□

NAMES

Names may contain as many as sixteen characters. The first character must be a letter, the following may be letters, digits, or the character "_". Some versions of COMAL do only allow letters and digits in names.

Example

```
NUMBER, MAX_NUMBER, NAME$, NAME_OF_PERSON$
```

□□□

CLOSING REMARKS

The line indentments on the listing are very important with COMAL, because they reflect the program structure. If an 80 character per line display is used, indentation will normally be two characters (see sample programs). If less than 80 characters per line is offered, the indentation may be reduced to one character. Normally it is not adviseable to use a line length of less than 64 characters with a COMAL interpreter. If, however, you have to do with less, it is of capital importance that broken lines are continued at the indentation level.

Example

If section 1140 - 1240 of the sample program "CASINO"

has to be listed on a 40 characters per line display or printer, it should look like this:

```
1140 PROC EXITGAMBLER
1150  PRINT
1160  IF ACCOUNT<>0 THEN
1170   PRINT "THE CONTENTS OF YOUR ACCOU
       NT, TOTAL $";ACCOUNT
1180   PRINT "IS RETURNED IN HARD CASH A
       T THE ENTRANCE."
1190  ENDIF
1200  PRINT
1210  PRINT "THANKS FOR THE GAME"
1220  IF WARNINGS<2 THEN PRINT "IT'S BEE
      N A PLEASURE."
1230  PRINT "COME BACK AGAIN SOME DAY"
1240 ENDPROC EXITGAMBLER
```

It may be that this is not an exquisite enjoyment to look at, but it does not totally ruin the structural lay out as is the case with certain barbarian printouts, where it is even allowed to have broken lines continued at the level of the line numbers!
If sufficient text processing facilities are available to the system editor, the listing above may even be moulded like this:

```
1140 PROC EXITGAMBLER
1150  PRINT
1160  IF ACCOUNT<>0 THEN
1170   PRINT "THE CONTENTS OF YOUR
       ACCOUNT, TOTAL $";ACCOUNT
1180   PRINT "IS RETURNED IN HARD CASH
       AT THE ENTRANCE."
1190  ENDIF
1200  PRINT
1210  PRINT "THANKS FOR THE GAME"
1220  IF WARNINGS<2 THEN PRINT "IT'S
      BEEN A PLEASURE."
1230  PRINT "COME BACK AGAIN SOME DAY"
1240 ENDPROC EXITGAMBLER
```

□□□

It should be pointed out that all COMAL versions have string arrays of at least one dimension. In most modern BASICs such arrays are available, so it will be no problem to have them in a COMAL that is built on such a BASIC. But if string arrays are not already available, they will have to be implemented in order that you get a running COMAL.

APPENDIX

The priority of the operators is

highest

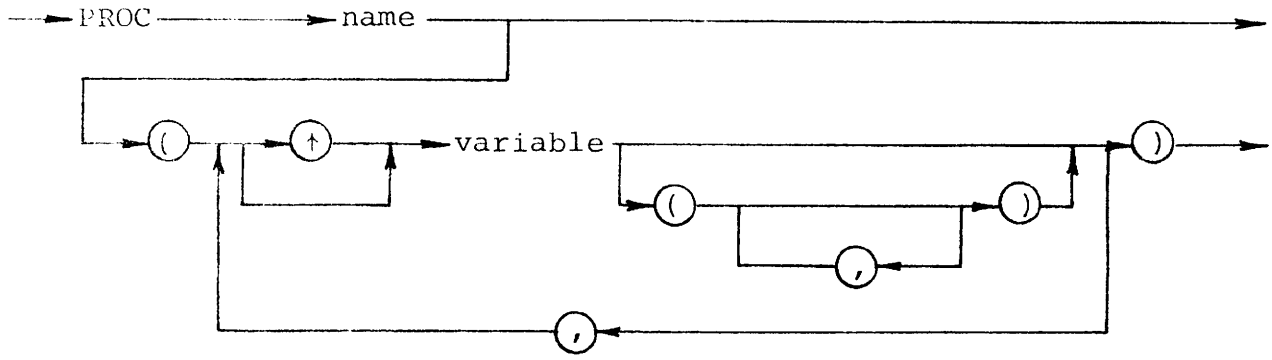
NOT

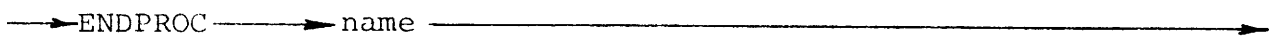
AND

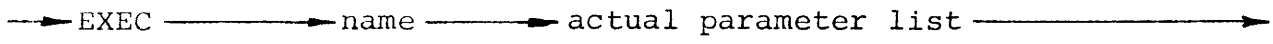
lowest OR

XCOMAL 80

SYNTAX DIAGRAMS







PARAMETERS

Parameters for procedures may be called by value or by reference. If a parameter name begins with the character ↑, it is called by reference and otherwise it is called by value. If an array is referred to by a parameter, its dimension must be indicated by means of parentheses and commas. Thus

```
↑NUM(,)
```

indicates a parameter that refers to a two dimensional array of numbers and is called by reference.

Example

```
PROC SORT(↑ACCOUNT(), MAX)
```

The first parameter must be called by reference and the second one must be called by value. The first one will refer to a one dimensional array of numbers, the second one will be assigned the value of a number.

□□□

Procedures may be called recursively to any depth as long as the main storage permits it.