
COMAL-80

A New Language?

Over the years, various magazines have published articles on the advantages and disadvantages of different languages.

BASIC has been the subject of many articles indicating its limitations, and pointing out that even a simple program seems rather difficult to read and understand.

The articles often conclude by expressing the need for a language which makes it natural to build up a program of smaller, logically linked parts, which can be called one by one in an orderly way.

One such language is Pascal, frequently mentioned in contrast to BASIC, and just as often described in emotional terms.

The silent war between Pascal and BASIC fans will probably never find its end, as some believe that Pascal does not have proper editing facilities (due to the construction as a compiler) and is too difficult to learn.

At this point it will be suitable to add that most persons taking part in the continuing discussions are EDP teachers showing an ever increasing interest for Pascal to match their higher levels.

Even before this war was started, an attempt to build a bridge took its start in Denmark. The teachers expressed their dissatisfaction with the possibilities at hand, and as constructive criticism always leads forward, a new language was implemented: COMAL-80.

This language has its family connections to both Pascal and BASIC, as it is believed to contain the best parts from both.

As mentioned above, the demand came from the educational sector asking for a better language, which is the reason for COMAL-80 having built-in programming support, and by its structure, facilitates a possible transition to more scientific languages. However, this does not mean that COMAL-80 is for education only, as it has proved valuable for administrative as well as controlling jobs, too.

It takes an extended BASIC and a good Pascal. Stir well and you get COMAL-80. At least, that is the way it may look to the users, but it certainly takes a lot more to take up the competi-

tion with established languages, benefiting from many years of adjustments to fit the market.

Rather, it takes a lot of planning, hard work, and precision to avoid loose ends to implement the language, meeting the high expectations shown in the preliminary specifications.

That the results far exceeded the expectations is proven by the fantastic response received from the more than 100 test-installations having been in use for about one year, before the current lead on the educational market in this country. The good results have spread the knowledge of COMAL-80 across the borders, and what seemed to be pioneer work within Denmark has led to many a plan for other high-level languages.

A Bit of History

The idea behind this language is not new. It started back in 1973 when Mr. Borge Christensen, assistant professor at the teachers college in Tondern, achieved daily access to a computer equipped with a BASIC – at the time apparently a good one.

Pretty soon he found several problems in running this BASIC. First of all, the students often had problems reading programs they had written themselves some time earlier. Among the reasons, Mr. Christensen found the awkward and unnecessary limitations for variable names, which was quite normal for BASICs at that time. Also, he found that in BASIC it is extremely difficult to write structured programs.

Mr. Christensen not only had the talent, but also the touch of luck needed to make the ideas come true. Acquiring the source-code was one thing, but having two excellent people in the area, Per Christiansen and Knud Christensen, who were able to improve the code, was just what he needed to get rid of the most obvious weaknesses.

The result was instant – students found it a lot easier to understand and work with this improved BASIC, and in the years following it was distributed to colleges and high schools teaching EDP.

In 1978 low-cost microcomputers really entered the market and EDP spread into grammar schools. However, the micros all had BASIC – and now the teachers had the knowledge of the *improved* BASIC.

The demand for better language on micros became so heavy that the imple-

mentation of a new language took its start in the summer of 1978. The first version was released for use on the COMET, marketed by International Computers Limited in March 1979.

Through the spring and summer this version was tested by more than 100 teachers and their students, whose comments and suggestions mostly turned into implemented improvements. Most of this work was done by Arne Christensen, a student of the Copenhagen University for Data Science, in close cooperation with Borge Christensen. (At this point it is worth mentioning that despite the similarity of their last names, the four gents are in no way related.) The product resulting from this work is marketed under the trade name of COMAL-80.

But What Is The Difference?

As described above, COMAL-80 appears as an improvement of BASIC. The most important aspect is that it works as an interpreter. Among the many important reasons for this, you will find that program errors can be dealt with very efficiently.

When a line is finished and the return-key is pressed, the error-message appears instantly. To continue the program, the error must be corrected. The cursor indicates the error-point and in many cases COMAL-80 suggests what needs to be done.

An example to this important feature is found in figure 1.

From BASIC another feature is inherited: the line numbers. It is incorporated in COMAL-80 because the basic idea is to be able to run any BASIC program, too.

It may seem strange to incorporate a full extended BASIC in a new language – rather like putting a Ford-T engine in an MG-sport – but the intention is very clear. Even programmers are slaves of routine and on numerous occasions it has proven extremely difficult to make them start all over on a new language. That is why COMAL-80 offers the programmer a chance to work as if it was the good old BASIC, and slowly move to the expanded facilities when the need occurs. Not only can one still be a BASIC-freak and run COMAL-80, but even the valuable programs already made in BASIC can run in COMAL-80. For the sake of honesty, BASIC has a lot of dialects and some of them are in fact contradictive; therefore some minor editorial correc-

by Susan Sonderstrup
and Mogens Pelle

tions may be needed, but certainly no routines need reprogramming.

Though the line-numbers are incorporated there is no need to use them, as all references to a program can be done by labels. There is no need to keep track of absolute references such as

```
30 GOTO 5000
```

although it can be done, it could as well be

```
30 GOTO start
```

for which "start" has been defined in another line, such as

```
100 LABEL start
```

Another significant difference which actually makes this language so powerful is the structured part which, as already mentioned, is taken in from Pascal.

First of all the procedure concept: this instruction is roughly an expansion of GOSUB in BASIC, but significantly more powerful. The instruction has the form of:

```
PROC name
```

```
:
```

```
ENDPROC name
```

name which may be called from any part of the program by the instruction

```
EXEC name
```

Parameters may be transferred to a procedure, which may even be declared closed. This gives the advantage of allowing the same variable names to be used both inside and outside a procedure. Such closing may again be opened by using the GLOBAL-instruction. Obscure procedure names are no longer needed, as all names may include up to 16 characters. Meaningful names improve the readability and the understanding of a program.

COMAL-80 includes conditional instructions. The BASIC instruction IF ... THEN is of course included, but it has been heavily expanded to the combined IF-instruction:

```
IF ... (THEN) ... (ELIF) ...  
(ELSE) ... (ENDIF)
```

In the same way

```
ON i GOTO k,l,m
```

may be used or substituted by a stronger term:

```
CASE ... (OF) ... WHEN ...  
(OTHERWISE) ... ENDCASE
```

Repeating instructions known from BASIC:

```
FOR ... TO ... NEXT
```

may even be used backwards in COMAL-80, replacing TO by DOWNTO. Further it includes 3 powerful instructions:

```
WHILE ... ENDWHILE
```

```
REPEAT ... UNTIL  
LOOP ... EXIT ... ENDLOOP
```

— instructions which certainly make working a lot easier to the programmer.

The above examples can give but an impression of the significant benefits included in COMAL-80. Hopefully, the program in Figure 2 will prove more than this brief description.

However, one more advantage deserves mentioning: the PRINT USING function has been expanded to have a certain resemblance to the FORMAT-instruction used in other languages, and it is allowed to write:

```
100 a$:="##.###"
```

```
200 PRINT USING a$: variable
```

How Does It Work?

As mentioned COMAL-80 has been implemented as an interpreter, the advantages of which are numerous, but leaving us with the drawback of lower speed. However, there are ways to compensate for this and increase the speed.

This program has been divided into three parts, each doing its own job.

Part one consists of two sub-modules: the editor and the syntax control working while keying in the program. The editor allows writing, deleting and inserting anywhere on the line, and upon completion, indicated by pressing RETURN, the line is transferred to the syntax-control, which checks the line against COMAL-80 specifications. If a syntax-error is found, the line is redisplayed, the cursor indicating the error-location. Also, an error-code and one of more than 200 error-messages, which include over 3,000 letters, explains what is wrong. At the same time it returns to the edit-mode, ready for corrections. This repeats until the line is accepted, upon which the line is transferred to internal format, ready for later execution.

Module 2 — the prepass — takes over when the program is to be executed. This module concludes the translation into internal format. Among others, it extends referring lines by the absolute memory address for the line referred to.

The program is now brought to a format in which the 3rd module can work very rapidly. In fact, the runtime module of COMAL-80 has proved in several cases to be faster than BASIC on a 16-bit minicomputer.

Availability

In March 1981, COMAL-80 was known to be available for different micros running the CP/M operating system, and by now it is probably available for other systems too.

Conclusion

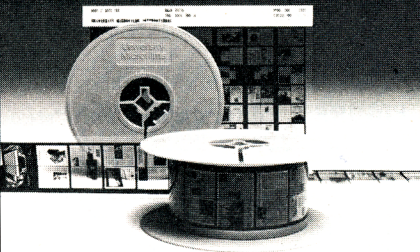
The implementation of COMAL-80 has demonstrated that a great idea may as well come from a small country, and that even the complicated task it is to turn the idea into a new EDP-language can be done to the extent of gaining international recognition.

For readers wanting further details on COMAL-80, they should contact Metanic ApS, Kongevejen 177, DK-2830 Virum, Denmark.



LISTING ON PAGE 46

This publication
is available
in microform.



University Microfilms
International

300 North Zeeb Road
Dept. P.R.
Ann Arbor, Mi. 48106
U.S.A.

30-32 Mortimer Street
Dept. P.R.
London WIN 7RA
England

Figure 1. Only the underlined words are keyed in by the user, whereas the rest shows how COMAL-80 by error-messages guides the user through to a correct program.

```
*10 DIM ARRAY<17
ERROR 14: ")" expected
10 DIM ARRAY<17
*20 OPEN
ERROR 30: "FILE" expected
20 OPEN FILE
ERROR 23: Operand expected
20 OPEN FILE 1
ERROR 21: ", " expected
20 OPEN FILE 1,
ERROR 23: Operand expected
20 OPEN FILE 1, "TEST"
ERROR 21: ", " expected
20 OPEN FILE 1, "TEST"
ERROR 43: READ/WRITE/RANDOM expected
20 OPEN FILE 1, "TEST", READ
*
```

end

Figure 2. More than words can tell... this program describes the strength of COMAL-80. Sorry the names are all in Danish.

```
0010 // POLYNY 27/11 - 1980
0020 INPUT "EPSILON: ": EPS
0030 DIM PROG$ OF 10
0040 DIM IND(0:50), POLY(-1:50)
0050 GR#:=0; ROD:=0; GROP#:=0
0060 MAT IND:=0
0070 MAT POLY:=0
0080 REPEAT
0090 INPUT "PROGRAM: ": PROG$
0100 CASE PROG$ OF
0110 WHEN "IND"
0120 EXEC IND
0130 WHEN "UD"
0140 EXEC UD
0150 WHEN "RETAB"
0160 EXEC RETAB
0170 WHEN "ROD"
0180 EXEC ROD
0190 WHEN "V#RDI"
0200 EXEC VERDI
```

```
0680 NEXT I#
0690 POLY(-1):=0
0700 ENDPROC RETAB
0710 PROC ROD CLOSED
0720 GLOBAL POLY, GR#, ROD, EPS
0730 INPUT "A,B: ": A, B
0740 FA:=FNA(A); FB:=FNA(B)
0750 IF FA=0 THEN
0760 M:=A
0770 ELIF FB=0 THEN
0780 M:=B
0790 ELIF SGN(FA)=SGN(FB) THEN
0800 PRINT "A: ";FA;" ";B: ";FB;"ENS FORTEGN!"
0810 ELSE
0820 IF FA>0 THEN A:=+B; B:=A-B; A:=-B
0830 ENDFI
0840 REPEAT
0850 M:=(A+B)/2; FM:=FNA(M)
0860 IF FM<0 THEN
0870 A:=M
0880 ELIF FM>0 THEN
0890 B:=M
0900 ELSE
0910 A:=M; B:=M
0920 ENDFI
0930 UNTIL ABS(B-A)<EPS
0940 PRINT M;" (";FM;")"
0950 ROD:=M
0960 ENDPROC ROD
0970 PROC VERDI CLOSED
0980 GLOBAL GR#, POLY
0990 DIM A$ OF 20
1000 REPEAT
1010 INPUT "START: ": A$
1020 SLUT:=A$=""
1030 IF SLUT THEN GOTO HOPUD
1040 FRA:=VAL(A$)
1050 INPUT "TIL: ": A$
1060 IF A$<"" THEN
1070 TIL:=VAL(A$)
1080 INPUT "STEP: ": A$
1090 IF A$="" THEN
1100 STP:=1
1110 ELSE
1120 STP:=VAL(A$)
1130 ENDFI
1140 ELSE
```

```

0210 WHEN "DIV"
0220 EXEC HORNER
0230 WHEN "DIF"
0240 EXEC DIF
0250 WHEN "KVA"
0260 EXEC KVA
0270 OTHERWISE
0280 //DO NOTHING
0290 ENDCASE
0300 UNTIL PROG#=" "
0310 PROC IND CLOSED
0320 GLOBAL IND, GR#, POLY, GROU#
0330 INPUT "GRAD: ": GR#
0340 GROU#:=GR#
0350 FOR I#:=GR# DOWNT0 0 DO
0360 PRINT USING "##: ": I#,
0370 INPUT " ": IND(I#)
0380 POLY(I#):=IND(I#)
0390 NEXT I#
0400 POLY(-1):=0
0410 ENDPROC IND
0420 PROC UD CLOSED
0430 GLOBAL POLY, GR#
0440 FOR I#:=GR# DOWNT0 0 DO
0450 PRINT USING "-##: ": I#,
0460 PRINT POLY(I#)
0470 NEXT I#
0480 PRINT "-----"
0490 PRINT "REST: ";POLY(-1)
0500 PRINT
0510 ENDPROC UD
0520 PROC HORNER CLOSED
0530 GLOBAL POLY, GR#, ROD
0540 X:=POLY(GR#)
0550 FOR I#:=GR#-1 DOWNT0 -1 DO
0560 Y:=POLY(I#); POLY(I#):=X; X:=X*ROD+Y
0570 NEXT I#
0580 POLY(GR#):=0; GR#:-1
0590 PRINT "REST: ",
0600 PRINT POLY(-1)
0610 PRINT
0620 ENDPROC HORNER
0630 PROC RETAB CLOSED
0640 GLOBAL IND, POLY, GR#, GROU#
0650 GROU#:=GROU#
0660 FOR I#:=GR# DOWNT0 0 DO
0670 POLY(I#):=IND(I#)
1150 TIL:=FRA; STP:=1
1160 ENDIF
1170 FOR X:=FRA TO TIL STEP STP DO
1180 PRINT X;" ";FNA(X)
1190 NEXT X
1200 LABEL HOPUD
1210 UNTIL SLUT
1220 ENDPROC VERDI
1230 DEF FNA(X)
1240 GLOBAL POLY, GR#
1250 Y:=POLY(GR#)
1260 FOR I#:=GR#-1 DOWNT0 0 DO
1270 Y:=Y*X+POLY(I#)
1280 NEXT I#
1290 FNA:=Y
1300 ENDEF FNA
1310 PROC DIF CLOSED
1320 GLOBAL POLY, GR#
1330 K1:=POLY(GR#)
1340 FOR I#:=GR# DOWNT0 1 DO
1350 K2:=K1; K1:=POLY(I#-1); POLY(I#-1):=K2*I#
1360 NEXT I#
1370 GR#:-1; POLY(-1):=0
1380 ENDPROC DIF
1390 PROC KVA CLOSED
1400 GLOBAL POLY, GR#
1410 IF GR#(>2 THEN
1420 PRINT "GRAD = ";GR#
1430 ELSE
1440 A:=POLY(2); B:=POLY(1); C:=POLY(0)
1450 D:=B*B-4*A*C
1460 IF D(<0 THEN
1470 PRINT "INGEN ROD. D = ";D
1480 ELSE
1490 IF B(<0 THEN
1500 X1:=(-B+SQR(D))/2/A
1510 ELSE
1520 X1:=(-B-SQR(D))/2/A
1530 ENDIF
1540 X2:=C/A/X1
1550 PRINT "X1, X2 = ";X1;" ";X2
1560 ENDIF
1570 ENDPROC KVA
1580
end

```