Modtaget 27/7-82 fra MP

Issue 3 - July 1982

CONTENTS

NEXT ISSUE

Next issue will be part two of the Uggley Story, more rumors and news, more program listings, programming tips, and hopefully an article from the creators of CBM COMAL on some of the advanced features of CBM COMAL. We welcome your contributions.

COMAL MADE IT

COMAL made it. Those three words say quite alot. It is the official language in the schools of both Denmark and Ireland, and soon Sweden and England. The four major companies providing COMAL interpreters have agreed upon a common standard called the COMAL KERNAL. It is supported by two newsletters, three books (and two more in the wings), and two users groups. It was the feature of a presentation at the National Computer Conference in the USA as well as featured at the Third International Commodore Computer Show. But possibly the best news, is that a new dual processor microcomputer system has been advertised as including COMAL as its standard language (BASIC does not come with the computer!). Aren't you glad that you can claim to be one of the COMAL pioneers! You had the foresight to see a major breakthrough in programming languages.

COMAL is THE official language

The education system in both Ireland and Denmark have adopted COMAL as the official language. Sweden and England may enjoy similar good fortune. Aparantly every school (except one) in Ireland now has a COMAL system. It has been that way in Denmark for several years.

COMAL has an official STANDARD

Up until May, 1982, there were several COMAL interpreters available. They each ran on different computer systems and were very similar to each other. However, they had some minor differences, causing incompatibility. These companies realized that it would be to their advantage, if a common COMAL KERNAL could be agreed upon. With this goal, there were many long meetings. The final meeting was held in Tonder, Denmark on May 7-9, 1982. It was

organized by Borge Christensen (the founder of the COMAL language). Instrutek, Denmark (CBM COMAL) was represented by Lars Laursen, Jens Erik Jensen, and Mogens Kjaer. Metanic Aps, Denmark (CP/M COMAL) was represented by Arne Christensen and Mogens Pelle. Regencentralen, Denmark (RC COMAL) was represented by Jorgen Olsen and Erik Steffensen. Trinity College, Ireland (TC COMAL), was represented by Colleen Kitchen and Kevin Ryan. Reston Publishing (The COMAL HANDBOOK) and the COMAL Users Group (USA) were represented by Len Lindsay.

Many compromises were made at this final meeting. Each group made concessions, thus the final COMAL KERNAL was not the least common denominator of all the different versions. Rather it emerged as a FULL COMAL, taking the best from each of the implementations. The standard COMAL is refered to as the COMAL KERNAL. If a language does not meet this KERNAL, then it is not a true full COMAL. Thus at the end of the meeting, there was no COMAL interpreter available that met the new KERNAL. However, each group agreed to update their versions to meet the new standard. Instrutek appears to be the first to finish this task. Their CBM COMAL version 1.02 should be available by the end of July 1982 for the CBM 8096 or as a plug in ROM board for any PET or CBM.

Two COMAL Newsletters

The original COMAL CATALYST (which you are reading) has been joined by the COMAL BULLETIN from England. The Bulletin is published by Ellis Horwood Limited, and is available for 10 pounds in England, 13.50 pounds overseas (about $30 US dollars?). The CATALYST subscription is free when you get the set of User Group Disks from the COMAL Users Group.

Three COMAL books

The original COMAL book is by Borge Christensen, founder of the language. COMAL PROBLEMLOSNING OG PROGRAMMERING is the original Danish version published by Bogika Data-Systemer. The German translation, COMAL 80, Die Strukturierte Sprache auf der Basis von BASIC, is published by Oldenbourg Verlag. The English translation, Beginning COMAL, is published by Ellis Horwood Limited.

STRUCTURED PROGRAMMING WITH COMAL by Roy Atherton, is published by Ellis Horwood Limited.

The COMAL HANDBOOK by Len Lindsay, is published by Reston Publishing (with a translation into Danish soon to be published by Bogika Data-Systemer). It is completely updated to meet the new COMAL KERNAL standard, which is printed in one of its appendices. The 300-400 page book includes over 100 complete sample programs and procedures.

Watch for two more books coming soon, one from Prentice-Hall, England, and the other from Sweden.

Two Users Groups

The original COMAL Users Group in the USA is an organization to help spread the good news about COMAL. The Users Group in England additionally has meetings. If you have a local COMAL group, please let us know. We offer 1 page in our newsletter to any COMAL group, for your news. The USA COMAL USERS GROUP has now been revitalized and will provide User Group Disks and a free newsletter. The only source of income is from the Disks (thus far advertising has been free). If you have any CBM COMAL programs you would like to place in the public domain, send them LISTed to disk to the COMAL User Group for inclusion on future User Group Disks.

Featured at the National Computer Conference

Kevin Ryan, from Ireland, presented COMAL at the NCC in Houston. The new COMAL interpreter from Trinity College was highlighted.

COMPUTER comes with COMAL as standard language

Full page, full color ads in Englands major microcomputer magazines announced the arrival of the GALAXY 1 COMPUTER by Gemini Microcomputers. This full featured micro system includes disks, keyboard, and monitor, as expected. But it is more advanced, and includes dual processors and COMAL-80 as part of its standard configuration.

REVIEW: COMAL ROM BOARD by Instrutek
Christianholmsgade, DK-8700 Horsens, DENMARK

With this board installed in your PET or CBM computer, you will have the full enhanced COMAL version 1.02. Whenever your computer is turned on it will come up in COMAL. To switch back to PET BASIC simply enter the word BASIC. The COMAL board supports both tape and disk, so an expensive disk drive is not required for operation. The board is a 64K ROM board, with 16 sockets for 4K EPROM chips. Ten of these sockets are used by the COMAL interpreter. The others are empty and available for your use. One 4K EPROM chip is available from Instrutek that allows you to control their high resolution graphics board using turtle graphic commands.

If you are concerned about getting this COMAL because it is not from Commodore, don't worry. Guess who wrote the COMAL interpreter for Commodore. Right. Instrutek. And they made sure that their board is completely compatible with the COMAL available for the CBM 8096.

CBM COMAL version 0.11 NOW OBSOLETE

CBM COMAL version 0.11 has been updated to meet the new COMAL KERNAL standard. The old version 0.11 is now obsolete and should not be used. Version 0.12, the update, is also in the public domain and can be freely copied. Note that it still is only an introductory COMAL and does not contain the complete COMAL KERNAL. CBM COMAL version 1.02 does contain the complete COMAL KERNAL, but will only run on the CBM 8096 (or it is available as a ROM BOARD to plug into any PET/CBM). Apparently CBM COMAL version 1.02 is also placed into the public domain, but this fact has not yet been verified. Major changes from version 0.11 to 0.12 include:

** Programs LISTed or EDITed to disk are now SEQ type files, an improvement from old version 0.11. These are compatible with the CBM 8096 version 1.02.
** ZONE is now a function.
Version 0.11-> ZONE:=2 and OLDZONE:=ZONE
Version 0.12-> ZONE 2 and OLDZONE:=ZONE
** Functions are now separate from procedures. Replace the PROC and ENDPROC statements with FUNC and ENDFUNC statements. The value of the function is now returned with a RETURN statement (RETURN <value>). The old version 0.11 assigned the value to the procedure name.
** Substrings are specified differently. The old version 0.11 used <start position>:<length> while the new version 0.12 uses <start position>:<end position>.
** The keyword DELETE has been added and is used to delete files from disk. **DELETE <filename>** is the syntax. The file name may contain wildcard patterns using the * and ? as in CBM DOS.
** The keyword PASS has been added. This is used to pass a command to the disk drive: PASS "<command>" --- example: PASS "i0" --- this initializes drive 0. Note that the dos 1.0 syntax is used for the commands.
** The keyword NULL has been added. It does absolutely nothing (a null statement).
** Several 'bugs' in version 0.11 have also been corrected.

TIP-FILE NAMES: When naming your files, take into consideration the following convention used by the COMAL Users Group. Files LISTed to disk should end with .L, BASIC programs should end with .B, Word Pro files with .W, and VisiCalc files with .V. This allows quick identification of program type from the directory listing.

NEWS-CBM 3.0 VERSION: A version of CBM COMAL version 0.11 has been converted to work with tape and BASIC 3.0 machines. However, version 0.11 is now obsolete, replaced by version 0.12 to meet the changes in the COMAL STANDARD at the May 1982 meeting in Denmark.

COMAL AT THE SHOW by Steve Kortendick

This season's feature event for Commodore fans in Europe was the Third International Commodore Computer Show held at the Cunard International Hotel in Hammersmith (London), England. Since 1980, the event has grown annually, to a point this year where the initially estimated space requirement of 18,000 square feet had to be supplemented by an additional 14,000 square foot exhibition hall. This is the extravaganza in which Commodore is king; in fact the entire show is fully dedicated to Commodore compatable hard- and software.

Among my greatest thrills at the show was stand 128. There, displaying three versions of CBM COMAL-80 were two of its creators: Jens Erik Jenson and Laurs Larson of Instrutek, the Danish firm commissioned by Commodore to write COMAL. With them was Brian Grainger, one of England's foremost supporters of the language.

This enthusiastic trio had several versions of COMAL on display, including a 3032 version, all in the CBM's RAM, an Instrutek ROM board version, and the 8096 RAM COMAL, with over 38,000 bytes of user memory left free. Also installed in one machine was the Instrutek add-on graphics board, the speed and response of which vastly superceded any single pixel addressable graphics option I have seen for home computer products. Further, all plot commands were directly addressable by procedure name from COMAL, making programming extremely straightforward.

It was a joy to meet those responsible for the version of COMAL I'm now using. Their enthusiasm for the language and their own enhancements, together with plans for future implementations was infectious. They're terribly interested in the public's reaction to their work, and appreciate both positive and negative feedback about the implementation, and ideas for future direction.

Of considerable interest among the many talks and seminars offered was the one by Brian Grainger of ICPUG (the famous Independent Commodore Product Users' Group). Brian's presentation was entitled "CBM COMAL-80: The Easiest First Lanugae to Learn," and covered the ease of use of COMAL, together with its logical yet simple structure.

One might expect a considerable amount of fanfare surrounding COMAL, the language which dares attack BASIC in the world of the micro. But few of the European show participants were terribly surprised. There, COMAL is a natural. It's clear that the national education boards of both Denmark and Ireland would not have mandated

the instruction of COMAL in the schools before the introduction of any other language without some familiarity being generated. To the Europeans, it is no longer the wave of the future, but the reality of today. I soon realized that COMAL information was available from several of the stands, ranging from local user groups to major educational displays. Whenever I inquired about COMAL, I received none of the "Oh sure, I've heard of 'COBOL'" responses so common here in the States.

Commodore, clearly number one in European small computers, is strongly backing the COMAL revolution. Rather than be left in the bitter dust of BASIC, Commodore has already commissioned CBM COMAL-80 and placed it in the public domain.

One high ranking official of Commodore UK confided that Commodore would soon be commissioning a software house to write a version of COMAL for the Commodore 64, VIC's new "big brother" being released this fall. This version will be in a ROM cartridge, plugging into the back of the "64," and targetted for school as well as home use. In fact, Commodore expects to sell these computers with COMAL cartridges in four-figure quantities to European school systems.

In Commodore's guide to the show, mention was made of The COMAL Handbook, by Len Lindsay. Unanimous among the COMAL supporters was the immediate need for that book to fill the information gap. Once this gap is bridged, there seems no bound to the challenge offered by COMAL to its ailing grandfather, BASIC. In Europe, COMAL is winning rapidly; we in the States can only hope for a similar outcome.

ABOUT THE AUTHOR: Steve is a programmer for the Department of Transportation, writing and maintaining programs to run on the huge AMDAHL mainframe computer system. He has had his own microcomputer for four years now, and currently is writing a small data base system in CBM COMAL version 1.02.

TIP-PROGRAM STRUCTURE SCAN: COMAL checks for correct syntax as you enter each line. However, it does not check for correct structures (an UNTIL for each REPEAT, etc.) until given the RUN command. So, to have your program scanned for correct structures, just make line 1 be: **0001 END.** A RUN will then check the structures without actually starting.

TIP-PROCEDUES: Use procedures and funtions as much as possible. Small modules are easier to debug and can be used in several programs without rewriting. CBM COMAL version 1.02 allows any of the procedures or functions to be called from direct mode after the program has been run. This is like adding new keywords to COMAL.

INFORMATION ABOUT CBM COMAL-80
by COMAL staff of INSTRUTEK

CBM COMAL-80 is a semi-compiler. When lines are entered from the keyboard, they are checked for syntax errors. During this parsing the lines are translated into an internal code (reverse polish). For example, the expression:

```
1 + 3 * SIN( - X / 2 ) + 3
```

is translated to reverse polish:

```
1 3 X 2 / - SIN * + 3 +
```

This allows fast program execution but also makes it harder to recreate (LIST) the original program lines.

When RUN is typed, a prepass is performed and the program is RUN. The prepass chains the program structures such as IF-ELIF-ELSE-ENDIF, REPEAT-UNTIL, etc. Absolute addresses are inserted in the program code. All structures are checked so that a REPEAT does not end with ENDIF and so on.

Because of the internal format of the program there are two types of program files:

A SAVE-file (PRG file) contains a prgram in the internal format. SAVE-files can be LOADed and CHAINed. It is useful because it is fast to load and is compact.

A LIST-file (SEQ file) contains the source text of a program. LIST-files can be ENTERed. It is useful when the program has to be transferred to different COMAL-80. It can also be used to handle procedure libraries.

CBM COMAL-80 version 0.12 works only on Commodore computers with 32K and BASIC 4.0, and leaves only 5-6K free memory for the user. It is meant as a free and introductory version. CBM COMAL-80 version 1.02 is an extended COMAL and can run with a CBM 8096 leaving 39K free memory. It also exists as a ROM version available from Instrutek which can be used with all CBM/PET computers with BASIC 2 or BASIC 4. It uses 2K of the RAM in the host CBM/PET. The CBM 8096 version is in the public domain.

About the AUTHORS:
The COMAL staff at Instrutek includes Jens Erik Jensen, Lars Laursen, Mogens Kjaer, and Helge Lassen. You would be hard pressed to find another group of such bright and dedicated young men. They are doing their best to make COMAL the very best language available. I think they are doing a marvelous job.

COMAL IS FAST

First we heard that the new language COMAL was structured and made programming and future modifications of programs easy. Next we heard that it was very "user friendly". Plus, disk loaded versions of COMAL for the the CBM 8032 and 8096 are public domain (you can make copies of the interpreter for all your friends). If that isn't enough to convert BASIC users, we put together a Benchmark Test System. We wrote the same system in CBM BASIC 4.0 and CBM COMAL version 1.02 (preliminary release). COMAL was faster than BASIC in **every** test (using a CBM 8032 with CBM 4040 disk). Both the BASIC and COMAL versions of the Benchmark Test System are available on a COMAL USERS GROUP disk. Next issue we hope to present both programs and the results. For now, a quick summary of the tests and how much faster COMAL was is shown below:

TEST TYPE / HOW MANY TIME FASTER COMAL IS	
Assign 1000 random coin flips to an array	/ 1.6
Classify 10 people by score 100 times	/ 2.7
Read 10 data strings 100 times (note: second group of data statements)	/ 9.3
Disk file write/reread numbers 1-1000	/ 1.8
Increment a variable 1000 times	/ 5.1
Loop 1000 times	/ 1.7
Process 10 choices of 6 items 100 times	/ 3.5
5 nested loops of 1 thru 10	/ 2.1
Assign a random die throw 1000 times	/ 1.4
Locate a string within another 1000 times	/55.2
Convert time from seconds to minutes & sec	/ 2.6
ALL THE TESTS	**/ 3.9**

COMAL did very well in processing choices (3.5 times faster than BASIC) and incrementing variables (5.1 times faster), both are important parts of most programs. And COMAL really was 55.2 times faster than BASIC in locating substrings (it is not a misprint, 55.2 times faster is correct). So if you compare strings alot, COMAL is ideal for you. COMAL also was more than 2 1/2 times faster in converting seconds into minutes and seconds. Although COMAL was almost 4 times faster for the combined tests, that result is not as significant as the specific tests.

TIP-CLOSED: Take advantage of the CLOSED option on procedures. A CLOSED procedure lets you use any variable names you wish without conflict with variables with the same name elsewhere in the program. You no longer have the variable name problem when adding a small routine to an existing program. In BASIC you probably used a wierd variable name like QW assuming it probably wasn't in use anywhere else. No such worry with COMAL.

THE UGGLEY STORY - by Borge Christensen

Mr. Otis Uggley, the general manager of the very large company, Uggley Publishers, always has difficulties finding the telephone numbers of the many members of his staff. The Uggley House has 14 departments, some of which has as many as 24 offices, and each office may have more than one local telephone number. At Uggley's a local telephone number has four digits, such as 2106. The two first digits is the number of the department plus 10, and the two last digits indicate the phone's number within the department. Thus the number 2106 shows that the telephone has been installed in deparment no. 21 as number 6. Mr. Uggley asks the company's house programmer, Mr. V. Cunning, to make it possible for him, Mr. Uggley, to retrieve the number of a member of the staff in an easy, fast, and safe way. Already the day after this request, Mr. Cunning has a small microcomputer installed besides Mr. Uggley's telephone. It is a new and very fine computer, using an efficient programming language called COMAL-80, he explains to Mr. Uggley, and a small program has been written to retrieve the telephone number of any member of the staff. Mr. Uggley only needs to enter the name of the person whose number he wants to know, then the computer will display the correct number in a split second.

Let's take a look at Mr. Cunning's problem and try to find out how he solved it so fast. "I shall do it fast and simple", he said to himself, "later on I may refine it or extend it to serve Mr. Uggley even better; right now we are very busy in this department!". To tell the reader the truth: they are always very busy right now in the programming department, and they always think that they shall have more time tomorrow, but that is a different story. His first program is shown above the next column.

The names of the staff members and their telephone numbers are set up in DATA statements to build a data queue. In the program below only a few persons and numbers are shown. Mr. Cunning used them for test purposes. Later on one of the typists at Mr. Uggleys secretariat typed in the full telephone directory.

The mainline part of the program is held in a REPEAT ... UNTIL loop. It starts in line 50 with a REPEAT statement and ends in line 170 with an UNTIL statement. It is obvious from the meaning of the keywords in plain English that the part of the program between the two statements is going to be executed repetitively until the variable THISNAME$ has assumed the empty string ("") as a value. The variable THISNAME$ gets its value from input, as is seen from the statement in 70: **INPUT "ENTER NAME: ": THISNAME$**

```
0010 PRINT CHR$(147) //CLEAR SCREEN
0020 //
0030 DIM THISNAME$ OF 30, NAME$ OF 30
0040 //
0050 REPEAT
0060   PRINT
0070   INPUT "ENTER NAME: ": THISNAME$
0080   RESTORE
0090   REPEAT
0100     READ NAME$,TEL
0110   UNTIL NAME$=THISNAME$ OR EOD
0120   IF THISNAME$=NAME$ THEN
0130     PRINT NAME$," HAS TELEPHONE NO. ",TEL
0140   ELSE
0150     PRINT THISNAME$," IS NOT ON MY LIST."
0160   ENDIF
0170 UNTIL THISNAME$=""
0180 PRINT CHR$(147) //CLEAR SCREEN
0190 PRINT "READY TO DO ANOTHER JOB....."
0200 //
0210 DATA "PETER BROWN",2406,"JOHN SMITH",1216
0220 DATA "ROY MANNING",1512,"ELIZA MOOR",1008
0230 DATA "MINNA JACOBS",1810
```

Each time this statement is executed, the text **ENTER NAME:** is output to prompt the user, and the program pauses to allow the user to type some string of characters, presumably the name of a person who is supposed to have a telephone, to be assigned as a value to THISNAME$. After the name of the person has been typed in, the program starts searching for the one in question in the data queue, using a very simple approach. First the RESTORE statement in line 80 resets the data pointer to indicate the first element in the queue, i.e. the name of the person recorded first, and after that the REPEAT ... UNTIL loop in line 90 - 110 takes over. The range of that loop holds only one statement:

```
READ NAME$,TEL
```

Each time this statement is executed a name and a number are retrieved, and the name is assigned as a value to NAME$, whereas the number is assigned to TEL. This is done repetitively,

```
UNTIL NAME$=THISNAME$ OR EOD
```

EOD is a Boolean function, i.e. a function that returns the values of TRUE or FALSE only. EOD is an abbreviation of "End Of Data". EOD returns the value of FALSE until the last element of the data queue has been read. Then it switches to return the value TRUE. The UNTIL statement above may be interpreted as "until the name retrieved from the queue is equal to the name entered or the queue is finished". Let us fancy that the name "ROY MANNINGS" is entered to become the value of THISNAME$. The name "PETER BROWN" is then assigned to NAME$ and the number 2406 to TEL. Since NAME$ ("PETER BROWN") is not equal to THISNAME$ ("ROY MANNING") and the end of the data queue has not been reached, line 100 is executed again, and NAME$ is assigned the value of "JOHN SMITH" and TEL the value 1216. The value of NAME$ is still not equal to the value of THISNAME$ and the End Of Data is FALSE, so line 100 is executed once more, this time to assign "ROY MANNING" to THISNAME$ and 1512 to TEL. And now the execution of the loop is terminated, because it is true that NAME$ is equal to THISNAME$; both have the value "ROY MANNING". The function EOD still returns a value of FALSE, but that does not matter because the Boolean operator OR connects the two Boolean expressions.

The searching has been sucessfully finished, and the program only needs to inform the user about the person's telephone number which is now the value of TEL. But take care! We have to consider the fact that it might have gone wrong! Two situations may occur: The person has been found, or the person has not been found on the list, and we had better be ready for both. This is done by means of the IF ... ELSE ... ENDIF segment in lines 120-160. The segment is controlled by the IF statement in line 120. If the Boolean expression following the keyword IF assumed a value of TRUE, the statement in line 130 is executed, but if a value of FALSE is returned, the statement in line 150 is executed instead. Now it is easy to see what happens in the case discussed above. Since it is TRUE that THISNAME$ is equal to NAME$ the statement:

```
PRINT NAME$," HAS TELEPHONE NO. ",TEL
```

is executed, and the following printout is produced:

```
ROY MANNING HAS TELEPHONE NO. 1512
```

Could you ask for more? Well, not from this program - that is, unless you enter a name of a person not in the telephone record!

If a name of a person, say "ROBERT BRADBEER", who is not in the telephone directory is entered, the expression **NAME$=THISNAME$** never returns the value TRUE, and the REPEAT ... UNTIL in line 90-110 is not stopped until EOD becomes TRUE, i.e. after all the names on the list have been looked at. It also implies that line 150 instead of line 130 is executed after the searching is done, with the following printout produced:

```
ROBERT BRADBEER IS NOT ON MY LIST
```

The program is stopped if the RETURN key alone is hit when the program asks for another name. This action assigns the empty string ("") as a value to THISNAME$ making the Boolean expression in the UNTIL statement in line 170 return a value of TRUE thus terminating the execution of the loop.

The example has demonstrated two of COMAL's fundamental structures: The REPEAT ... UNTIL loop and the IF ... ELSE ... ENDIF double branching. The REPEAT ... UNTIL loop always works according to this scheme:

```
REPEAT
  ---------------------
   block of statements
  ---------------------
UNTIL <Boolean expression>
```

The structure is working straightforward: The block of statements appearing between the REPEAT and the UNTIL statement - the range of the loop - is executed repetitively until the Boolean expression in the UNTIL statement returns a value of TRUE.

The IF ... ELSE ... ENDIF structure works according to this scheme:

```
IF <Boolean expression> THEN
  ----------------------
   block of statements-1
  ----------------------
ELSE
  ----------------------
   block of statements-2
  ----------------------
ENDIF
```

If the Boolean expression in the IF statement returns a value of TRUE, the first block of statements is executed, and processing continues with the statement following the ENDIF. If, however, the Boolean expression evaluates to FALSE, only the second block of statements is executed, and processing then continues with the statement following the ENDIF. The ELSE part can be left out to give the following:

```
IF <Boolean expression> THEN
  ---------------------
   block of statements
  ---------------------
ENDIF
```

A block of statements may hold as many statements as you want, and the statements may be of any kind. The indentation of the lines in a block in the program listing is done automatically by the interpreter.

The program uses long variable names: THISNAME$, NAME$, TEL. In COMAL variable names may consist of up to 16 letters or digits, though the first character must be a letter (editors note: CBM COMAL allows names of up to 78 characters). All characters are significant, such that for example NAMEOFMAN$ and NAMEOFMEN$ are two different variables in COMAL - though it would hardly be adviseable to use them in the same program!

As mentioned before, if Mr. Uggley wants to stop looking numbers up, he should do nothing but strike the RETURN key, when the program comes out asking for another name. The trouble is that after that, the whole searching affair is carried through, with the program virtually searching for nothing. Of course the Boolean expression **NAME$=THISNAME$** returns a value of FALSE in the end, because persons with names that cannot be mentioned are not hired to work in The Uggley House. This state of affairs makes the statement in line 150 produce the output:

```
 IS NOT ON MY LIST
```

This printout is actually produced, but is hardly noticed by the user, because it is wiped away a split second later by the statement in line 180 which clears the screen. It is not nice, but it works, and as mentioned before, Mr. Cunning was very busy! Mr. Cunning is, however, also a very good programmer, and he does not like to have a "dirty" program running in The Old Man's office. A few days later he uses an opportunity to modify the program to become:

```
0010 PRINT CHR$(147) //CLEAR SCREEN
0020 //
0030 DIM THISNAME$ OF 30, NAME$ OF 30,
0040 //
0050 INPUT "ENTER NAME: ": THISNAME$
0060 WHILE THISNAME$<>"" DO
0070   RESTORE
0080   REPEAT
0090     READ NAME$,TEL
0100   UNTIL NAME$=THISNAME$ OR EOD
0110   IF THISNAME$=NAME$ THEN
0120     PRINT NAME$," HAS TELEPHONE NO. ",TEL
0130   ELSE
0140     PRINT THISNAME$," IS NOT ON MY LIST."
0150   ENDIF
0160   PRINT
0170   INPUT "ENTER NAME: ": THISNAME$
0180 ENDWHILE
0190 PRINT CHR$(147) //CLEAR SCREEN
0200 PRINT "READY TO DO ANOTHER JOB....."
0210 //
0220 DATA "PETER BROWN",2406,"JOHN SMITH",1216
0230 DATA "ROY MANNING",1512,"ELIZA MOOR",1008
0240 DATA "MINNA JACOBS",1810
```

As you can see, the REPEAT ... UNTIL loop to control the mainline program has been replaced by another structure that begins with a WHILE statement in line 60 and ends with an ENDWHILE statement in line 180. A WHILE ... ENDWHILE structure is also a loop structure, only this time the Boolean expression to control the loop is sitting at the beginning of the loop - in the WHILE statement - and not at the end of the loop as it did before.

The INPUT statement to take in the value of THISNAME$ has been replaced by two of the kind, one in line 50 and one in line 170. When the program is started the statement in line 50 comes out asking for a name. As soon as a string has been typed in, the WHILE statement is executed, and if the expression **THISNAME$<>""** returns the value TRUE, the block of statements in the range of WHILE ... ENDWHILE is executed, i.e. if a non empty name is entered the search and printout is carried out as before. When the statement in line 170 is executed, another name is asked for, and when that request has been answered the execution continues with the WHILE statement. If the name typed in is not the empty string, the interior of the loop is executed again, and this goes on as long as no empty string is entered. However, as soon as the RETURN key alone is pressed and the empty string is assigned to THISNAME$, the Boolean expression in the WHILE statement returns the value FALSE - THISNAME$ is no longer different from the empty string - execution of the loop is stopped, and processing continued with the statement following the ENDWHILE.

This time Mr. Cunning has done his homework. Because the test is now sitting at the entrance of the loop, its range is not executed after the empty string has been entered and assigned to THISNAME$. But it has cost one extra INPUT statement. Good structure sometimes has its price.

The WHILE ... ENDWHILE loop structure works according to the following scheme:

```
WHILE <Boolean expression> DO
  ---------------------
    block of statements
  ---------------------
ENDWHILE
```

The block of statements between the WHILE and the ENDWHILE statements - the range of the loop - is executed repetitively as long as, i.e. while, the Boolean expression in the WHILE statement returns a value of TRUE. If it returns a value of FALSE, execution of the loop is stopped and continued with the statement following the ENDWHILE. You should notice that if the Boolean expression returns a value of FALSE right from the beginning, the range of the loop is not executed at all. You may have as many statements in the block as you like and they may be of any type. The indentation on the listing of the statements in the block is generated automatically by the COMAL system. The short REPEAT ... UNTIL loop including lines 80-100 may of course be replaced by a WHILE ... ENDWHILE loop, too. It would look like this:

```
0080   WHILE NOT (NAME$=THISNAME$ OR EOD) DO
0090     READ NAME$,TEL
0100   ENDWHILE
```

Since there is only one statement between the WHILE and the ENDWHILE statements, the whole loop may be written on one line:

```
WHILE <Boolean expression> DO <statement>

0080   WHILE NOT
   (NAME$=THISNAME$ OR EOD) DO READ NAME$,TEL
```

This construct explains why we have the keyword DO to terminate the WHILE statement. During the following two weeks Mr. Cunning continues to "brush up" his program, and finally ends up with this version (including test data only):

```
0010 PRINT CHR$(147) //CLEAR SCREEN
0020 //
0030 DIM THISNAME$ OF 30, NAME$ OF 30,
0040 //
0050 INPUT "ENTER NAME: ": THISNAME$
0060 WHILE THISNAME$<>"" DO
0070   //
0080   //LOOK FOR IT:
0090   RESTORE
0100   WHILE NOT (NAME$=THISNAME$ OR EOD) DO↘
0105   //                    \READ NAME$,TEL
0110   //DISPLAY MESSAGE:
0120   IF THISNAME$=NAME$ THEN
0130     PRINT NAME$," HAS TELEPHONE NO. ",TEL
0140   ELSE
0150     PRINT THISNAME$," IS NOT ON MY LIST."
0160   ENDIF
0170   //
0180   //GET ANOTHER NAME:
0190   PRINT
0200   INPUT "ENTER NAME: ": THISNAME$
0210 ENDWHILE
0220 //
0230 //DONE:
0240 PRINT CHR$(147) //CLEAR SCREEN
0250 PRINT "READY TO DO ANOTHER JOB....."
0260 //
0270 DATA "PETER BROWN",2406,"JOHN SMITH",1216
0280 DATA "ROY MANNING",1512,"ELIZA MOOR",1008
0290 DATA "MINNA JACOBS",1810
```

You may have noticed the symbol // to appear in several places in the program. It is used to indicate a comment (like REM in BASIC). Since a comment may be empty the symbol can be used as a kind of "segment separator", and this is how it is used in line 20, 40, 220 and 260 in the latest version of Mr. Cunning's program (editors note: CBM COMAL allows a line to completely blank, without even the //). However, in line 10 and 240 you can see that a comment may also be placed after any statement.

One morning, about a fortnight after Mr. Cunning had finished the program, Mr. Uggley called him on the phone. "I have some problems with this new system of yours", he said. "yesterday I had a report sent up, and I wanted to ask for some details after having read it", he went on, "but the only reference to the person who had written it, was the signature which said P. BROWN. I entered the name "P. BROWN" in that new telephone thing, as you have instructed me to do, but the imbecile answered nothing but "P. BROWN IS NOT ON MY LIST". Now would you be kind enough to tell me, what this is all about?".

Of course Mr. Cunning knew what was wrong. "Oh, I am a fool. Why did I not tell him to let IT do it in the first place. It is most likely that he will go on for ever now asking for more, more, more ...!", he sighed, and went upstairs.

Mr. Cunning changed two statements in one of the earlier versions of his program to get the following:

```
0010 PRINT CHR$(147) //CLEAR SCREEN
0020 //
0030 DIM THISNAME$ OF 30, NAME$ OF 30,
0040 //
0050 INPUT "ENTER NAME: ": THISNAME$
0060 WHILE THISNAME$<>"" DO
0070   RESTORE
0080   REPEAT
0090     READ NAME$,TEL
0100   UNTIL THISNAME$ IN NAME$ OR EOD
0110   IF THISNAME$ IN NAME$ THEN
0120     PRINT NAME$," HAS TELEPHONE NO. ",TEL
0130   ELSE
0140     PRINT THISNAME$," IS NOT ON MY LIST."
0150   ENDIF
0160   PRINT
0170   INPUT "ENTER NAME: ": THISNAME$
0180 ENDWHILE
0190 PRINT CHR$(147) //CLEAR SCREEN
0200 PRINT "READY TO DO ANOTHER JOB....."
0210 //
0220 DATA "PETER BROWN",2406,"JOHN SMITH",1216
0230 DATA "ROY MANNING",1512,"ELIZA MOOR",1408
0240 DATA "MINNA JACOBS",1810,"PAUL BROWN",1205
0250 DATA "MARY JOHNSON",2402,"MARY BROWN",1101
```

Then he instructed Mr. Uggley to enter whatever information he had about the name of the person he was looking for, i.e., "BROWN" or "JACOBS" or the like, but never to use abbreviations as in "P. BROWN". "My program cannot cope with that", he said. Mr. Uggley looked a little disappointed, but after having entered the name "BROWN" and watched the answer "PETER BROWN HAS TELEPHONE NO. 2406" come up on the screen he was satisfied. Mr. Cunning left his office with a sigh of relief, knowing far too well that ...

In the meantime let us take a look at the program. The statements in line 100 and 110 have been changed. The statement in line 100 used to say:

```
UNTIL NAME$=THISNAME$ OR EOD
```

but now it says:

```
UNTIL THISNAME$ IN NAME$ OR EOD
```

The Boolean expression **THISNAME$ IN NAME$** uses the relational operator "IN" which is found in COMAL-80. Its effect is extremely simple: If the value of the first string variable is a substring of the value of the second variable, the expression returns a value of TRUE, otherwise it returns a value of FALSE. Now it is easy to see what happened when Mr. Uggley typed in the word "BROWN". First it was assigned to THISNAME$ as usual. After NAME$ had assumed the value "PETER BROWN", the expression **THISNAME$ IN NAME$** evaluated to TRUE since "BROWN" is in fact a substring of - or found in - the string "PETER BROWN". It is also obvious that entering the string "P. BROWN" would not work, because "P. BROWN" is not a part of "PETER BROWN" when we use simple pattern matching.

The relational operator **IN** is defined in the set of string as follows:

```
<string expression-1> IN <string expression-2>
```

The Boolean expression thus given evaluates to TRUE if the string returned by the first expression is a substring of the value returned by the second expression; otherwise a value of FALSE is resulting. The operator **IN** does, however, give more than that. In case the first string is a substring of the second one, an expression like the one above does not only return a value different from zero - and is therefore interpreted as equivalent to TRUE - but it gives the position of the first string in the second. Thus with THISNAME$ equal to "BROWN" and NAME$ equal to "PETER BROWN" the expression **THISNAME$ IN NAME$** returns the value of 7, because "BROWN" begins at position number 7 in

"PETER BROWN". In that way two birds are killed with one stone. **IN** can be used both as a relational operator and a positional operator.

Very much to his surprise, Mr. Cunning hears nothing from Mr. Uggley during the next few days. He goes over the program and there is one small detail that does not quite satisfy him. It is not to his taste that the somewhat longish expression **THISNAME$ IN NAME$** appears twice. It also slows down the program because it is computed eah time it is encountered. Furthermore it is very easy to remedy, and of course Mr. Cunning knows how. He changes the segment in line 80-110 to become:

```
0080   REPEAT
0090     READ NAME$,TEL
0100     FOUND:=THISNAME$ IN NAME$
0110   UNTIL FOUND OR EOD
```

Let us take a look at one of the new statements he has introduced:

```
FOUND:=THISNAME$ IN NAME$
```

As we know already, the Boolean expression to the right of the assignment returns a numeric value. It evaluates to zero if THISNAME$ is not a substring of NAME$ but to the position - i.e. a number different from zero - of the first string in the second if it is there. Whatever value the expression returns is assigned to FOUND thus making it possible for this variable to bear witness where ever needed of the outcome of the computation.

It is needed in the statement in line 110:

```
UNTIL FOUND OR EOD
```

which replaces the statement in old line 100:

```
UNTIL THISNAME$ IN NAME$ OR EOD
```

This can be done because FOUND has been assigned the value of the expression **THISNAME$ IN NAME$** and thus "knows the truth about it". The variable FOUND also replaces the same Boolean expression in the statement: **IF FOUND THEN.**

It is no longer necessary to evaluate the expression one extra time because FOUND still remembers the outcome.

When a numeric variable is used like FOUND in the above example, it is also called a Boolean variable. Some peope prefer to call it a flag, because it is used as a signal to the rest of the program about the outcome of some test.

The reason why Mr. Cunning had not heard anything from Mr. Uggley appeared to be that the latter had been away on business in Paris - that is at least what he had told his wife - but now he was back! Only half an hour later he rang Mr. Cunning, obviously quite upset about something. "This last program of yours is even worse than the old one!", he yelled. "Oh, he's found out!", Mr. Cunning murmured to himself, but Mr. Uggley did not hear that, "I asked for Mr. Brown this morning and got Peter Brown as last time. Only this time he appeared to be the wrong person! And know what? I've got three Browns in this house: Peter Brown, Paul Brown, and Mary Brown, and you cannot expect a man in my position to keep in check any Peter, Paul, or Mary strolling around here. When I ask for Brown, I want that silly device you have installed here to give me the names and numbers of all the Browns in this house! ALL! Do you understand?". Oh, yes, Mr. Cunning had understood all this long ago, and furthermore he was ready to meet the challenge.

He had solved that problem several days ago and jotted it down on a piece of paper. This is what his paper says:

```
...
repeat
  repeat
    get a name and telephone number
    found the name?
  until found or end of data
  if found then
    printout name and number
  endif
until end of data
if not found then
  printout that the name is not on the list
endif
get another name
...
```

The main problem he had to cope with is that searching cannot be stopped just because one person whose name matches with the name that has been type in has been found. There may be more. Fancy the name "BROWN" has been entered. Then it is not enough that "PETER BROWN" is found; "PAUL BROWN" and "MARY BROWN" must be picked up too. If a name has been found but end of data has not been reached, the program must go on searching. This is obtained by introducing an extra loop that forces the searching to go on until end of data is finally reached. Each time a name to match the entry has been found, it is displayed with the telephone number that goes with it. Only if the search has been totally unsuccessful should the user be told that no name in the telephone directory has any resemblance whatsoever with the name entered.

But this raises one problem. If a Boolean variable like FOUND in the program above is used to flag sucesses as well as failures, it is of course set to TRUE each time a matching name has been found, but in any other case it is set to FALSE. Unless the name we are looking for is the last one on the list, FOUND will always end up with a value of FALSE, even though several names to match may have been found earlier!

The problem can be solved by introducing another flag of a more persistent nature than FOUND. We shall use a "black flag", i.e. one which holds a value of TRUE - is set - as long as no name has been found, but is assigned a value of FALSE - is reset - if only one name is found and displayed. Mr. Cunning's algorithm now comes to look like this:

```
...
none:=true // maybe none of that name
repeat
  repeat
    get a name and number
    found the name?
  until found or end of data
  if found then
    printout name and number
    none:=false // at least one
  endif
until end of data
if none then
  printout that name is not on list
endif
get another name
...
```

And now it is your turn, dear reader. Use Mr. Cunnings notes and parts of the other programs he has written to set up a COMAL-80 program that will solve Mr. Uggley's problem - and Mr. Cunning's! A solution will be suggested in the next article. (Editors note: Next issue will hold the next installment of the Uggley story)

ABOUT THE AUTHOR:

Borge Christensen is the founder of the COMAL language and author of the book BEGINNING COMAL. He is the principal lecturer for Mathematics and Computing at the College of Higher Educaton in Tonder, Denmark. If you like the style of this article please let us know. We are encouraging him to write a "COMAL STORYBOOK".

RUMOR-COMMODORE 64: Commodore hopes to have a COMAL ROM PACK for their new COMMODORE 64 computer. Just plug the cartridge in and you have a full COMAL computer. Apparantly they are hoping to provide the Irish school system with 1000 - 4000 systems (including the COMAL pack for less than the list price without the pack).

TIP-VERSION 0.11 LIST FILES FIX: CBM COMAL version 0.11 makes its LIST files type PRG instead of SEQ. Thus a program LISTed to disk in version 0.11 cannot be entered by version 0.12 or 1.02 due to a file type mismatch. The FIX procedure will read a PRG type LIST file created by version 0.11 from drive 0 adn write it to drive 1 as an SEQ type file. It then can be ENTERed into version 0.12 and 1.02 COMAL. To use FIX, simply RUN it. Then type **FIX("NAME")** - replace NAME with the name of the file. Make sure that the version 0.11 LIST file is in drive 0 and a disk is in drive 1 for the new version. If you have alot of programs to convert - TRANSFER will convert all the PRG files in drive 0 to SEQ files in drive 1. It uses FIX, so include both in your program. Once RUN simply type **TRANSFER**. Procedures FIX & TRANSFER are listed below:

```
PROC TRANSFER CLOSED
  IMPORT FIX                    4040 Drive only
  DIM SKIP$ OF 1, FILE'NAME$ OF 16
  PASS "IO"
  OPEN FILE 7,"$0",READ
  SKIP$:=GET$(7,162); SKIP$:=GET$(7,92)
  BLOCK'COUNT:=0
  WHILE NOT EOF(7) DO
    FILE'TYPE:=ORD(GET$(7,1))
    SKIP$:=GET$(7,2)
    FILE'NAME$:=GET$(7,16)
    SKIP$:=GET$(7,11)
    BLOCK'COUNT:+1
    IF BLOCK'COUNT MOD 8 THEN SKIP$:=GET$(7,2)
    IF FILE'TYPE=130 THEN FIX(FILE'NAME$)
  ENDWHILE
  CLOSE FILE 7
ENDPROC TRANSFER

PROC FIX(OLD'FILE$) CLOSED
  DIM NEW'FILE$ OF 20, LINE$ OF 100
  DIM NEW'DRIVE$ OF 2, OLD'DRIVE$ OF 2
  DIM NEW'TYPE$ OF 4, OLD'TYPE$ OF 4
  NEW'DRIVE$:="1:"; OLD'DRIVE$:="0:"
  NEW'TYPE$:=",SEQ"; OLD'TYPE$:=",PRG"
  NEW'FILE$:=OLD'FILE$
  PRINT "FIXING:";OLD'FILE$
  OPEN FILE 2,OLD'DRIVE$+OLD'FILE$+OLD'TYPE$,READ
  OPEN FILE 3,NEW'DRIVE$+NEW'FILE$+NEW'TYPE$,WRITE
  WHILE NOT EOF(2) DO
    INPUT FILE 2: LINE$
    PRINT FILE 3: LINE$
  ENDWHILE
  CLOSE FILE 3
  CLOSE FILE 2
ENDPROC FIX
```

NEWS-APPLE: If you have an APPLE II+ with a language card and disk drive, you can get a version of COMAL for the system. Apparently the Irish school system bought 500 of these systems.

USER GROUP DISKS FINALLY

Now that the CUG is not distributing COMAL STARTER KITS, the distribution of User Group Disks is virtually our only income (the newsletter remains free). Thus far most COMAL programs in our library are either from DENMARK or Madison, WI. Only one other person has sent us a disk of sample programs so far. Our current problem is that CBM COMAL has just changed to meet the new COMAL KERNAL. Thus most of our programs must be updated accordingly. I have written several programs to aid in the conversion process. These aids plus all programs we can convert will be put onto master disks and should be available by the time you read this. The disks should be useful to you, as well as extremely informative as to how COMAL programs are put together. **Most important, each disk will contain both new version of CBM COMAL, version 0.12 and version 1.02.** The disks are public domain and may be freely copied. The programs will run on either version 1.02 or 0.12 or both. The programs will be stored on the disk as LIST-files (SEQ files). They then can be ENTERed into either version. If you have only version 0.12, you can list the programs that can run only with version 1.02 from disk to printer easily, so that you can analyze the program, to see if you can adapt it. FORMATTER will do this elegantly. A simple way to do it is with the following program:

```
DIM LINE$ OF 100
OPEN FILE 2,<program name>,READ
SELECT "LP"
WHILE NOT EOF(2) DO
  INPUT FILE 2: LINE$
  PRINT LINE$
ENDWHILE
SELECT "DS"
CLOSE
```

All programs on the User Disks and their grouping is subject to change without notice, but here is how we plan to procede:

**** UTILITY DISK 1:**

FORMATTER::: List programs with variable width, lines per page, indentation amount, left margin, and page heading with page numbers. Programs can be listed with or without line numbers.

SCAN & DISK'SCAN::: Aid in conversion of old version 0.11 programs to new version 0.12. SCAN reads a LIST file and lists lines that may be of concern, such as assigning a value to a procedure name (the old way of performing a function) and possible substring use (the substring specification has been changed). DISK'SCAN will do a whole disk, one file at a time, unattended.

FIX & TRANSFER::: Version 0.11 LIST files are a type PRG and cannot be entered into version 0.12 or 1.02 which expect SEQ type LIST files. FIX reads a file from drive 0 and writes it to drive 1 as a SEQ type LIST file. TRANSFER will do the whole disk, one file at a time, unattended.

DIRECTORY::: Another way to solve the PRG / SEQ file type conflict. It actually changes the PRG in the directory to be SEQ. It shows you how the directory can be modified by a COMAL program.

REMOVE //::: Removes all remarks in a program.

READ'DIR::: Reads the disk directory.

AUTO'MENU::: Creates a menu of all PRG files on the disk with a fast moving cursor controlled pointer. Point to the program you want to run, and hit return.

AUTO'RUN.B:::A COMAL Loader program that will LOAD the COMAL interpreter and then automatically RUN a COMAL program from disk. A good way to have the AUTO'MENU run.

FILE'TO'SCREEN::: For use with the Instrutek high resolution graphics board.

DIFF:::

DOS:::A very good DOS interface similar to the WEDGE used by BASIC.

SYNTAX1 & SYNTAX2::: Simulates the COMAL interpreter by checking the syntax of anyline you type.

NAME'TABLE::: Lists all variables, arrays, procedures, and functions used by a program.

RECREATE::: Lists a program from its SAVE file.

DISK::: Disk command interface.

EXPRESSION::: Evaluates expressions.

PACK & UNPACK::: For use with the Instrutek high resolution graphics board.

BASICREADCOMAL.B:::A BASIC program example to read a COMAL data file.

WPTOCOMAL.B::: Convert a Word Pro file to a COMAL file.

BASICLISTCOMAL.B:::A BASIC program that will list a COMAL program LISTed to disk.

HEX'TO'DECIMAL::: Conversion program.

**** APPLICATION DISK**

UGGLEY SERIES PROGRAMS::: The telephone directory programs that go along with the continuing article by Borge Christensen.

BALLCLUB PROGRAMS::: The programs needed to maintain a random data base of members of a ball club (or any other group).

MONEY TRACKER PROGRAMS::: Track where you money goes by account number via sequential disk files.

ORDER PROCESSING PROGRAMS::: Process orders and keep a disk file of all customers.

OLSEN MAIN PROGRAMS::: Converted from version 0.11 to 0.12.

ENGLISH LESSONS:::

SUBTRACTION AID:::

HISTOGRAM::: Creates a histogram from your data, and includes a movable pointer to point to smallest and largest points, etc.

TURTLEGRAPHICS PROGRAMS::: For use with the Instrutek high resolution board.

DISK MANAGEMENT SYSTEM::: Keeps track of all your programs and disks. Not yet completed.

**** DEMO & FUN DISK**

OTHELLO

LABYRINTH::: You are inside a maze. Rather than show you the maze, it shows you your view as if you were walking down a corridor. Nicely done. Set of three programs.

POLYGON PUZZLE::: With data files and graphic display.

TOWERS OF HANOI::: Well done display of the solution to this classic problem.

MAGIC SQUARE:::

SILLY:::

QUEENS::: Solution to the queens chess problem.

COMAL LOGO:::

MUSIK:::

QUICKSORT:::A fast sort routine that you can use with your programs.

CLOCK:::

**** VARIOUS PROGRAMS**

There are several dozen more programs not mentioned that will hopefully end up on one disk or another. We welcome your program subissions for future disks.

CBM COMAL DISK COMMANDS

```
Scratch File: DELETE <filename>
  DELETE "0:TEMP*"
 or in 0.12: PASS "S<drive>:<filename>"
  PASS "S0:STARTER.L"
  PASS "S0:TEMP2,CODES,MY3"  (several files)
Directory: CAT [<drive>] [<pattern>]
  CAT 1 "*=seq"   (only seq files listed)
 or in 0.12: CAT [<drive>]
  CAT
Initialize: PASS "I[<drive>]"
  PASS "I0"
Format/New: PASS "N<drive>:<diskname>,<id>"
  PASS "N0:DISK,D8"
Reformat: PASS "N<drive>:<diskname>"
  PASS "N1:DISK"
Collect/Validate: PASS "V<drive>"
  PASS "V0"
Backup/Duplicate: PASS "D<destination>=<source>"
  PASS "D1=0"   (backup drive 0 to drive 1)
Copy: PASS "C<destination>=<source>"
  PASS "C0=1"   (copy drive 1 to drive 0)
Copy file: PASS "C<dst>:<newfil>=<src>:<oldfil>"
  PASS "C0:MENU.FINAL=1:MENU.PRELIM"
Rename file: PASS "R<drive>:<newname>=<oldname>"
  PASS "R0:CREATE'REC=TESTING"
```

TIP-USE LIST FILES: IMPORTANT: Each version of CBM COMAL uses a different LOAD / SAVE program format. However, the LIST to disk or tape is always the same. Therefore, whenever you wish to transfer a program from one version to another LIST it to disk, then ENTER it from the new version. Whenever sending programs to other users, always LIST them to disk or tape so that any version of COMAL can retrieve them.

UPDATE TO HANDBOOK IN STARTER KIT

The COMAL STARTER KIT is no longer being distributed by the COMAL USERS GROUP due to the transfer of the rights of the COMAL HANDBOOK to RESTON PUBLISHING, who will be publishing a much expanded version in August 1982 (300-400 pages!!!). The expanded version covers CBM versions 0.11, 0.12, and 1.02. However, here are some updates to the old HANDBOOK:

pg 5: The IN operation returns a <numeric expression>
pg 62: If the string supplied to ORD is more than one character, it only looks at the first character.
pg 64: The word OUTPUT is optional and if left out will be supplied by the system.
pg 65: The sample RUN output should not include the letters R U N.
pg 69: The syntax for the OPEN statement can end with **[,UNIT <dev>[,<sec adr>]][,<type>]**
pg 60 & 93: UNIT has a bug in the interpreter, and at times will yield a syntax error when it shouldn't. It is corrected in version 0.12. The correct syntax is:

```
OPEN [FILE] <num>,<name>[,UNIT <dev>[,<secadr>]]
  [,<type>]
  <num> is the file number
  <name> is the file name
  <type> is READ, WRITE, APPEND or
            RANDOM <rec len>
  <rec len> is the record length
Examples: OPEN FILE 2,"TEST",UNIT 9,READ
          OPEN FILE 3,"",UNIT 3
```

TIP-READ PRG FILES: The standard OPEN statement in CBM COMAL will allow you to read sequential (SEQ type) files. It is possible to read other file types as well by including the file type as the last part of the file name preceded by a comma: <filename> = <drive>:<name>,<type>. For example: "0:MY'PROGRAM,PRG".

UPDATE TO CBM COMAL MANUAL

The following are updates recieved on the CBM COMAL MANUAL, called A SHORT SURVEY OF CBM COMAL-80 in the COMAL Starter Kit. The CASE structure was accidently omitted and is printed here. Also, any additions made to the survey concerning COMAL version 0.12 (update from 0.11) will be listed as found in the updated survey.

CASE, WHEN, OTHERWISE, ENDCASE

The CASE statement is the head of the CASE structure, that controls multiway branching. The syntax of a CASE statement is:

```
CASE <expression> [OF]
```

<expression> is a numerical, string or boolean expression. The keyword OF is bracketed to indicate that it will be supplied automatically by the COMAL interpreter if not typed in by the user. Control is performed by the CASE statement only if it is assisted by the WHEN and ENDCASE statements. The syntax of the WHEN statement is:

WHEN <list of expressions>

The expressions on the list following WHEN must be of the same type as the expression in the master CASE statement. The expressions on the list must be separated by commas. The ENDCASE statement consists only of one word: **ENDCASE.** There may also be an OTHERWISE statement in the CASE structure. Its syntax is only one word: **OTHERWISE.** The CASE structure works according to this diagram:

```
CASE <expression> [OF]
WHEN <list of expressions>
  <statements-A1>
WHEN <list of expressions>
  <statements-A2>
...
WHEN <list of expressions>
  <statements-An>
OTHERWISE
  <statements-B>
ENDCASE
```

When the expression following CASE has been evaluated, the list following the first WHEN is examined. If one of the expressions on this list has the same value as the CASE expression, program section A1 is executed and control is then passed to the statement following the ENDCASE statement. If no such item is found, the list following the second WHEN is examined. If the value of the expression is found, A2 is executed, and control is then passed to the statement following ENDCASE. If the value still has not been found, the interpreter starts on the third list, etc.

A default case (statements-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 opened with the OTHERWISE statement. The OTHERWISE case may be left out but the interpreter will then stop the execution of the program with an error message if the value of the expression following CASE has not been found in the WHEN lists. Note that at most one of the cases is executed. If it so happens that the value of the expression may be found in more than one of the lists, only the first of these lists will trigger off its process. The program texts of A1, A2, ..., An, and B are indented in the program listing. This indentation is supplied automatically by the COMAL interpreter.

Examples: Sample program "AUNTIE", lines 710-830 and 1630-1750 and sample program "OLSENMAIN", lines 130-310. Many other examples to be found in sample programs.

DATA

As in BASIC, DATA statements are used to hold numeric or string constants that may be retrieved in a READ statement. The individual constants are separated by commas, and string constants must be contained between quotation marks.

END

The END statement makes the computer terminate execution of a program.

ENDWHILE

The ENDWHILE statement is used to terminate the block of statements controlled by a WHILE statement. See WHILE.

ESC

The keyword ESC is used in two different ways: As part of a TRAP statement (see TRAP), and as a standard function (see STANDARD FUNCTIONS).

EXPRESSIONS

In CBM COMAL-80 you have arithmetic expressions and string expressions. Arithmetic expressions can contain constants, variables, and functions, used with parentheses and the following operators according to the usual rules of mathematics:

```
 +   monadic + (+A)
 -   monadic - (-A)
 ^   power (A^B)
 *   multiplication (A*B)
 /   division (A/B)
DIV  integer division (A DIV B) see DIV
MOD  remainder from division (A MOD B) see MOD
 +   addition (A+B)
 -   subtraction (A-B)
```

Numeric values or strings may be compared by means of the following relational operators:

```
  <   <=   =   >=   >   <>
  IN   used for string pattern matching. see IN
  NOT  logical negation. NOT A returns a value of
       FALSE if A is true, but a value of TRUE if
       A is false.
  AND  logical conjunction. A AND B returns a
       value of TRUE if a A and B are both true
  OR   logical disjuncation. A OR B returns a
       value of FALSE if A and B are both false
```

NOTE In CBM COMAL-80 a numeric value equal to zero is interpretted as FALSE, whereas any value different from zero is interpreted as TRUE. A logical operation returns a numeric 1 for TRUE and 0 for FALSE.

The priority of the operators is:

```
  ^
  * / DIV MOD
  + -
  < <= = >= > <> IN
  NOT
  AND
  OR
```

A string expression may consist of string constants, string variables, or string array elements concatenated by means of the + sign.

LIST (add to page)

A program that has been stored by means of LIST "<name>" can be opened as a normal ASCII file by using: **OPEN 2,"<name>",READ.** The individual lines of the file may now be retrieved by using a statement like: **INPUT FILE 2: LINE$.**

NEW - clears whole workspace of program & data

POKE

POKE I,J is a statement that assigns a value of J to byte no. I in the computers memory.

PRINT (add to page)

In a PRINT statement the TAB functions may be used to set the next print position. The statement: **PRINT 1,TAB(20),2** displays the constant 1 in the first column of the line and the constant 2 in column 20 of the same line. If the argument of the TAB function evaluates to a position before the current one, the line is shifted first. If the argument is less than or equal to 0 an error message is given.

RESTORE - resets the data pointer to indicate the first element in the data queue

---

RETURN

If used in a function it may have this syntax:
RETURN <arithmetic expression>
The value of the expression will then be returned by the function.

RUN

The RUN command invokes a prepass of the program (all structures are linked and structural errors are reported) and then starts execution of it.

SETEXEC

The command **SETEXEC-** makes the interpreter suppress the keyword EXEC when listing a program. The command **SETEXEC+** resets the interpreter to display the keyword EXEC. On the start of the COMAL system SETEXEC- is executed automatically. When typing in a program, the keyword EXEC is optional.

STANDARD FUNCTIONS

ABS(X) - returns the absolute value of X
ATN(X) - returns the arctangent in radians of X
CHR$(X) - returns the character whose ASCII value is equal to X
COS(X) - returns the cosine of X (X in radians)
EOD - returns a value of TRUE if the last element in a DATA queue has been read, otherwise a value of FALSE is returned
EOF(X) - returns a value of TRUE if the end-of-file mark in a sequential file, opened with channel no. X, has been retrieved, otherwise a value of FALSE is returned
ESC - returns a value of TRUE if the STOP key has been depressed, otherwise it returns a value of FALSE. The ESC function is not active unless a TRAP ESC- statement has been encountered.
EXP(X) - returns the value of e to the power of X
INT(X) - returns the integer part of X, i.e. the greatest integer less than or equal to X.
LEN(X$) - returns the current length of the string value of X$
LOG(X) - returns the natural logarithm of X
ORD(X$) - returns the ASCII value of the first character held by X$
SGN(X) - returns the sign of X: -1 if X is negative, 0 if X is 0, 1 if X is positive
SQR(X) - returns the square root of X
TAN(X) - returns the tangent of X (X in radians)

STOP

A STOP statement stops the execution of the program. Execution can be resumed from the statement following the STOP statement by means of the CON command.

SYS

The statement **SYS I** causes the interpreter to make a subroutine call (JSR) to byte no. I in the computers memory.

TRAP

A TRAP statement or command is used to disable the normal functioning of the STOP key. After the statement or command **TRAP ESC-** has been encountered by the interpreter, depressing the STOP key will not as normal result in the program execution being stopped, but instead the system variable ESC will be set to TRUE. The normal reaction of the system on the STOP key is restored by the command or statement **TRAP ESC+.**

TIP-ROUNDING: Assigning a real value (like 1.6) to an integer variable (like NUM#) results in the value being ROUNDED before assignment. This makes an easy way to round a number rather than truncate it with the INT function.

```
FUNC ROUND(NUM) CLOSED
  N#:=NUM
  RETURN N#
ENDFUNC ROUND
```

PRESENTING AN AUTOMATIC MENU IN COMAL

Automatic menu selection of programs is a fantastic aid. A utility to provide this function should be in everyone's library of programs. In the past 5 years, I have written several such menu programs. First, one for the PET with CompuThink disk drive. Then one for the Atari 800. Next one for the IBM 3033 (a huge mainframe). Now, the best one yet. The following menu presents a menu of only the PRG type files, and provides a very fast cursor (pointer) to slide around. Just hit <return> when it points to the program you want to run. The program demonstrates how to read the CBM 4040 directory and how to move a pointer.

```
// AUTOMATIC MENU / PROGRAM SUBMISSION
// BY LEN LINDSAY - JULY 1982 - 80 COL
DIM NAME$ OF 17, MOVE$ OF 1
ZONE 0
CLOSE
OPEN FILE 3,"",UNIT 3,3
INTRO
DIRECTORY
POINTER

PROC POINTER
 ROW:=1; COL:=1; POINT:=0
 REPEAT
  MOVE$:=KEY$
  CASE MOVE$ OF
  WHEN CHR$(19) // HOME
   POINT:=0
  WHEN CHR$(145) // UP
   POINT:=-1
  WHEN CHR$(17) // DOWN
   POINT:=+1
  WHEN CHR$(157) // LEFT
   POINT:=-24
  WHEN CHR$(29) // RIGHT
   POINT:=+24
  WHEN CHR$(13) // RETURN
   DO'IT
  OTHERWISE
   NULL
  ENDCASE  // CHR$(146) IS REVERSE OFF
  PRINT AT ROW,COL: CHR$(146)," ", //ERASE THE >
  IF POINT>=FILE'COUNT THEN POINT:=FILE'COUNT-1
  IF POINT>=24*4 THEN POINT:=24*4-1
  IF POINT<0 THEN POINT:=0
  ROW:=1+(POINT MOD 24)
  COL:=1+20*(POINT DIV 24)
  PRINT AT ROW,COL: CHR$(18),">",
 UNTIL FALSE=TRUE // CHR$(18) IS REVERSE ON
ENDPROC POINTER

PROC INTRO
 PRINT CHR$(147),CHR$(18), // CLEAR SCREEN/REVERSE ON
 CURSOR 25,1
 PRINT "  USE CURSOR KEYS TO POINT TO PROGRAM TO RUN.",
 PRINT " HIT <RETURN> TO RUN THE >PROGRAM.",
ENDPROC INTRO

PROC DIRECTORY
 DIM SKIP$ OF 1, FILE'NAME$ OF 16
 PASS "IO"
 OPEN FILE 7,"$0",READ // for 4040 DRIVE ONLY
 SKIP$:=GET$(7,162); SKIP$:=GET$(7,92)
 BLOCK'COUNT:=0; FILE'COUNT:=0
 WHILE NOT EOF(7) DO
  FILE'TYPE:=ORD(GET$(7,1))
  SKIP$:=GET$(7,2)
  FILE'NAME$:=GET$(7,16)
  SKIP$:=GET$(7,11)
  BLOCK'COUNT:+1
  IF BLOCK'COUNT MOD 8 THEN SKIP$:=GET$(7,2)
  IF FILE'TYPE=130 THEN
   IF FILE'COUNT<24*4 THEN
    ROW:=1+(FILE'COUNT MOD 24)
    COL:=2+20*(FILE'COUNT DIV 24)
    PRINT AT ROW,COL: FILE'NAME$,
    FILE'COUNT:+1
   ENDIF
  ENDIF
 ENDWHILE
 CLOSE FILE 7
ENDPROC DIRECTORY

PROC DO'IT
 NAME$:=GET$(3,16) // NAME
 CLOSE
 PRINT AT 25,1: " LOADING";NAME$;".....",
 PRINT ".....",
 CHAIN NAME$
ENDPROC DO'IT
```

ABOUT THE USERS GROUP & EDITOR

Hi! As CBM COMAL moves into its second year, the COMAL Users Group is still alive and kicking. The CUG is run by only one person, and being that person, I can verify that it is alot of work. I could handle to work, if I had the time. But there's the catch. I have two 'real' jobs. I work as technical writer for a "microprocessor specialists" consulting firm during the day, and evenings (till midnight) I run an IBM 3033 computer system for the State of Wisconsin. With the two jobs, there is not much spare time left over (the spare time also was used to complete my third book for RESTON PUBLISHING). Since I can't do everything, I now have two people helping me with the newsletter and User Group Disks. Two other people have started a small company, COMAL INTEREST GROUP, to distribute COMAL books, programs and interpreters. (NOTE: I am not part of this new group). I am trying to find a publisher for the newsletter that can take care of the subscriptions, layout, printing and distribution. I still will take care of getting the material for it. I also would like someone to organize the many programs we are collecting onto User Group Disk masters. The many different versions of COMAL make this a problem. Right now we are only supporting CBM COMAL with the CBM 4040 disk format.

YOU MAY NOT WANT TO HEAR THIS BUT...

You don't need me to tell you what others have been saying for years. Computers are becoming more advanced every year. After all the advances and innovations in the computer hardware over the past 20 years, why are so many people still using BASIC, which is a rather primitive language compared with some of the new ones available now. Well, until last year I could answer that easily. BASIC came with the computer, it was FREE. It was easy to learn and use. The other better languages may be better for professional programmers, but not for the ordinary non-professional home programmer. They were too rigid, required you to understand complex principles and usually were not interactive. Languages were paraded around us, each claiming superiority. PILOT claimed to be easier to use (but lacked some of the power). FORTH claimed to be fast, and indeed it is. If you need application software for industrial control, FORTH may be for you. Then APL appeared for the microcomputer. You would see a letter from an APL programmer showing a one line program (with triangles and squiggles) that would do some mathmatical calculation that BASIC would need 20 lines to do. But who could read it? And how about PASCAL and C, both somewhat similar. C is good for people who don't like to type, since it use brackets alot to save typing in BEGIN etc. But it is not too readable. PASCAL is readable but imposes too many rules and regulations on you. EVERY variable must be declared. Besides just declaring it, you must specify what type of variable it is, REAL, INTEGER, etc. And each line must end with a semi-colon (with some exceptions). The program must have the word END. at the end (including the period). For some reason, PASCAL can't figure out where the end of the program is. PASCAL is also very bad in string handling and Input/ Output. Possibly the worst part is that PASCAL has no line numbers and requires the use of an EDITOR that is separate from PASCAL. So you must learn to use the EDITOR as well as learn to use PASCAL.

I tried PILOT, FORTH, and PASCAL. I found that PILOT and FORTH were not suited to my situation. PASCAL would have been nice, but it got me too frustrated with all its silly rules.

"If you have been used to great freedom [with BASIC], you will come to regard your PASCAL compiler as a strict and fussy schoolmaster. You still have the chance to be creative, certainly, and you can still have fun, but you must play strictly by the rules." --- page 18, PASCAL FROM BASIC by Peter Brown.

It has been said that COMAL has the ease of BASIC with the power of PASCAL. The corollary to this is that COMAL doesn't have the problems of BASIC nor the problems of PASCAL. (I use CBM COMAL-80 version 1.02 for reference) If you are considering leaving BASIC behind, read the book PASCAL FROM BASIC by Peter Brown. It is very good. It shows how BASIC can be improved with structure, but also points out PASCAL's shortcomings. After reading it you will not want to use either language (BASIC or PASCAL). Then remember that COMAL doesn't have to problems of PASCAL, some of which are indicated by two quotes below from the book:

"Modern education is often held to emphasize flamboyant and trendy subjects at the expense of basic skills. PASCAL suffers in the same way. It is strong on data structures and the like but comparatively weak on the three R's: reading, riting, and rithmetic. Its rithmetic lacks an exponentiation operator, and, an omission felt by commercial programmers, decimal operation. Its reading and riting, i.e. its input and output, seem to be designed to help sell Bill's BASIC FROM PASCAL book. It is not so much that PASCAL's input/output is short on facilities; it is just that some fundamental things are difficult or impossible to do."

"You have probably got the impression, after reading this Chapter, that getting anything in or out of your PASCAL program is like going round the Royal Saint George's golf course equipped only with a putter."

PASCAL FROM BASIC is available from Addison-Wesley Publishing Company for $12.95. Copyright 1982.

So there I was (as many of you may be right now), knowing full well that BASIC was outdated, and not being able to find a suitable replacement. PASCAL is not meant to be a replacement for BASIC for the home computer. But in May of 1981 there was a major breakthrough for the PET/CBM. COMAL arrived on the scene like a ray from heaven. COMAL is an advanced programming language for the non-professional programmer which includes the advantages of PASCAL without losing the friendliness of BASIC. Any one who uses the FULL ENHANCED COMAL (CBM COMAL version 1.02) for a week, will not want to go back to BASIC. (Version 1.02 is available on disk for the CBM 8096, or as a plug in board from Instrutek for any PET/CBM except the original PET with BASIC 1.0 --- introductory version 0.12 is available on disk for any PET/CBM with BASIC 4.0. All of this is available thru the COMAL USERS GROUP).

You may have heard about COMAL before. You heard that it allows structured programming and even indents the structures automatically so you can SEE them (PASCAL does not provide this 'pretty printing' automatically. You must run a program whose listing is 7 pages long to list your program nicely). You may already know that COMAL allows a multiple line IF ... THEN ... ELSE. You may know that it has both a REPEAT ... UNTIL and a WHILE ... ENDWHILE loop along with the standard FOR ... NEXT. You have been told that it has the wonderful CASE structure (replacing the ON ... GOTO). Then finally you probably have been told about COMAL's multi-line FUNCTIONS and PROCEDURES, both allowing parameter passing and local or global variables. So you already know that COMAL is far superior to BASIC. But many probably still have not used COMAL. Why? Is it fear of change?

Well, fear no more. The COMAL system is on your side, helping you, not fighting you with a bunch of silly rules. COMAL uses line numbers simply for your use in editing the program. The line numbers are not used by the program itself. You can delete blocks of lines with one command (DEL). The system will prompt line numbers for you automatically with the AUTO command. A renumber command (RENUM) is available if you need it. You can list all or part of your program. The listing automatically indents the structures (pretty printing). To LIST a specific procedure, INTRO for example, simply type, LIST INTRO. A program can be listed without the indenting if you wish. Simply add a file name to the LIST command, and the program, or program segment, will be listed to disk or tape. These segments can later be merged into another program via the ENTER command. The disk directory can be printed in full, selectively using pattern matching, or even just listing the SEQ files (or just PRG or USR files).

If you have a printer, it can be turned on or off with the SELECT command. Everything works the same on the printer as on the screen, including TAB and ZONE. So once you have your program output looking nice on the screen, simply add the line SELECT "LP" (for Line Printer) and the program will print the same thing, formatted nicely, just as it was on the screen. (You can't easily do this with BASIC). COMAL provides you with two different types of sequential files, as well as direct access (random) disk files. Standard PASCAL has only a sequential file capability. COMAL has the GET$ and KEY$ statements, greatly improved over BASIC's simple GET. COMAL lets you easily 'tack on' your own enhancement set of 'keywords'. Just define them as procedures or functions. Once a program is RUN all procedure and function names are remembered by the system, and can be called from direct mode at any time (the sky is the limit here). For example, lets say you have defined a function in your program called GCD (greatest common divisor) that had two parameters (the two numbers to test for the greatest common divisor). After you are done running the program, COMAL still remembers that function. You can now find the greatest common divisor of any two numbers in direct mode. For example: PRINT GCD(35,21) --- COMAL will reply with 7.

COMAL has PRINT USING, allowing the formatting of numbers into neat columns, as well as PRINT AT, allowing you to specify the row and column to begin printing on the screen. It reads DATA statements with a READ statement (Did you know that standard PASCAL does not have this capability!). It also has the RESTORE statement like BASIC, but with a major improvement. You can restore the next DATA pointer to any line you wish. But the line is not specified by its line number (remember, line numbers are not significant to a running COMAL program). Instead you include a label on the line before the DATA line to be restored to, and use RESTORE NAME (using the name of the label of course). BASIC has a 'tab' point every 10 columns on the screen and when you print items separated by a comma, BASIC skips to the next column when it comes to the comma. COMAL calls these tab positions ZONEs, and lets you assign them to be whatever interval you wish. They will work on the printer as well as the screen. So if you wish to have columns 6 characters wide simply say ZONE 6. COMAL also has the CURSOR statement that will put the cursor in whatever ROW and COLUMN you wish. Text can be read right off the screen under program control too. Plus division with an integer answer is provided for with the DIV statement. A MOD statement is also available, giving the remainder to the division. You can disable the STOP key if you want. The words TRUE and FALSE are part of COMAL (just like in PASCAL). PEEK and POKE are permitted, as well as SYS (so you still can directly play with the system). Machine code can be loaded by a program with the command OBJLOAD. Programs can be CHAINed together without any problems (BASIC has a few problems in this area). String arrays are available, and can have up to 33 dimensions each with whatever top and bottom indexes you wish to use (more than 256 is no problem). Of course numeric arrays also have this capability, but allow up to 36 dimensions.

Variable names in BASIC are a problem since only the first two characters are significant. COMAL allows variable names to be up to 78 characters long, with all the characters significant. These characters can include the apostrophe ('), square brackets ([]), backslash (\), and underline (represented by the left arrow key),

along with the 26 letters and 10 digits. The backslash and square brackets are included so that languages with some extra 'letters' can use these. Danish uses one of these for the AE combination letter. These long variable names apply to both strings (end with a $) and integer (end with a #).

Substrings are very easy with COMAL. No need to use special words like MID$ to manipulate a string. You can print part of a string by specifying its start character position and its end character position (BASIC would use RIGHT$, LEFT$, and MID$ to do this). But COMAL also allows you to change a substring to something else without affecting the rest of the string (BASIC won't let you do this).

COMAL also includes end of data (EOD) and end of file (EOF) pointers, making reading in data from data statements or files very easy.

Those were just **some** of the advantages of the language. Now I will mention how COMAL is a FRIENDLY language. It often will know what you want but not require you to type it (i.e., PASCAL requires an END. statement in every program). BASIC is friendly in this respect, but COMAL is even more friendly. Both allow you the option of leaving off the closing quote mark of a string constant if it is at the end of a line. But COMAL does more than allow you to skip it, it puts it in for you so that your line lists with the quote mark at the end. COMAL distinguishes between the comparison equals and the assignment equals (as does PASCAL) by use of a colon in front of the equal sign ':=' to mean assignment. But you don't have to type the colon, for COMAL will put it in for you. And incrementing a variable is easier than with BASIC (borrowing from ALGOL). In BASIC, to add a number to the total you would say: T=T+N while in COMAL you would say: T:+N or more appropriately, TOTAL:+NUMBER. This feature is available with strings as well as numbers.

CBM COMAL provides the same full screen editor that all PET/CBM users are fond of (and that most other micros don't have). In addition to that, COMAL checks each line as it is entered for correct syntax. If it finds a mistake, it prints a very helpful error message under the line, and puts the cursor at the spot in the line of the suspected error. Simply correct the error and hit RETURN. The error message is erased from the screen and the line that it overwrote is restored (the error message is therefore non destructive to the information on the screen). This is a fantastic system, especially the first few times you use COMAL.

COMAL often will provide keywords you leave out if it knows that they belong in the statement.

For instance, to select the printer for your output you can type SELECT "LP", but COMAL will insert the word OUTPUT and list the line as SELECT OUTPUT "LP". The word THEN (in the multi-line IF ... THEN statements) can be omitted and will be supplied by COMAL.

So, what language should you now be using? That depends upon your situation. Both the Danish and Irish school systems are now using and teaching COMAL as their official language (with Sweden and England soon to follow). I haven't used BASIC for over a year now and haven't suffered from any withdrawl symptoms. I enjoy using COMAL. It makes you feel like a great programmer.

RESOURCE LIST

*Bogika Data-Systemer, Akacieparken 38, DK-7430 Ikast, Denmark. Tel. (07) 153155

*Borge Christensen, States Training College, Ostergade 65, DK-6270 Denmark

*COMAL Interest Group, 505 Conklin Place, Madison, WI 53703

*COMAL Users Group, 5501 Groveland Ter, Madison, WI 53716 USA. Tel. (608) 222-4432

*COMAL Users Group, John Collins, 4 Grimthorte House, Percival Street, London, EC1V 0B5, England

*Ellis Horwood Limited, Market Cross House, Cooper Steet, Chichester, West Sussex, PO19 1EB, England

*Gemini Microcomputers, Oakfield Corner, Sycamore Road, Amersham, Bucks HP6 5EQ, England

*Instrutek, Christiansholmsgade, DK 8700, Horsens, Denmark. Tel. 05 61 1100

*IPUG, Mick Ryan, 164 Chesterfield Drive, Riverhead, Sevenoaks, Kent, England

*Metanic Aps, Kongeveien 177, 2830 Virum, Denmark. Tel. (02) 858284

*Oldenbourg Verlag, Rosenheimer Strasse 145, 8000 Munchen 80 W Germany

*Regnecentralen, Lautrubjerrg 1-5, DK 2750, Ballerup, Denmark. Tel. (02) 658000

*Reston Publishing, 11480 Sunset Hills Road, Reston, VA 22090 USA. Tel. (703) 437-8900

*Trinity College, Department of Computer Science, Dublin 2, Ireland

CONVERT FROM CBM COMAL VERSION 0.11 TO 0.12

To transfer a program written with CBM COMAL version 0.11, first LIST the program to disk (i.e., LIST "MY'PROGRAM"). Next convert the file into a SEQ type file. Use program DIRECTORY (on Utility User Group Disk) or FIX (listed on page 11). Next scan the program for statements that need to be changed do to the update to the new COMAL KERNAL. The program SCAN below prints lines with statements that use a procedure as a function (change the PROC and ENDPROC to FUNC and ENDFUNC, and the value must be returned with RETURN). It also prints lines that may need changing due to substring specification changes. To call SCAN simply type **SCAN("MY'PROGRAM").**

```
PROC SCAN(FILE'NAME$) CLOSED
 DIM LINE$ OF 100, PROC'NAME$ OF 80
 PRINT FILE'NAME$;":"
 OPEN FILE 4,"O:"+FILE'NAME$,READ
 PROC'NAME$:=""; IN'PROC:=FALSE
 WHILE NOT EOF(4) DO
  INPUT FILE 4: LINE$
  IF IN'PROC THEN
   IF " ENDPROC "+PROC'NAME$ IN LINE$ THEN
    IN'PROC:=FALSE
   ELSE
    FLAG1:="$(" IN LINE$
    IF PROC'NAME$+":=" IN LINE$ THEN
     PRINT ":= ";LINE$
    ELIF " RETURN " IN LINE$ THEN
     PRINT "RET";LINE$
    ELIF FLAG1 THEN
     IF ":" IN LINE$(FLAG1:LEN(LINE$)) THEN PRINT "$(:";LINE$
     IF "," IN LINE$(FLAG1:LEN(LINE$)) THEN PRINT "$(,";LINE$
    ENDIF
   ENDIF
  ELSE
   PROC'POS:=" PROC " IN LINE$
   IF PROC'POS THEN
    IN'PROC:=TRUE
    END'NAME1:=" " IN LINE$(PROC'POS+6:LEN(LINE$))
    END'NAME2:="(" IN LINE$(PROC'POS+6:LEN(LINE$))
    IF END'NAME1=0 THEN
     PROC'NAME$:=LINE$(PROC'POS+6:LEN(LINE$))
    ELIF END'NAME2 AND END'NAME1>END'NAME2 THEN
     PROC'NAME$:=LINE$(PROC'POS+6:PROC'POS+5+END'NAME2-1)
    ELSE
     PROC'NAME$:=LINE$(PROC'POS+6:PROC'POS+5+END'NAME1-1)
    ENDIF
    //PRINT "PROC'NAME=";PROC'NAME$
   ENDIF
  ENDIF
 ENDWHILE
 PRINT
 CLOSE FILE 4
ENDPROC SCAN
```


TIP-ENTERING A DIFFERENT VERSION FILE: When ENTERing a program from a different COMAL version, you may encounter some SYNTAX ERRORS or such. If this happens, the offending line will be displayed on the screen with an accompanying message. Either fix the problem, or if you wish, simply move the cursor to the beginning of the line and insert a // (or a !) just after the line number (makes the line a remark) and hit RETURN. The system continues to enter the rest of the program. After the whole program is entered, you may then go back to those lines and correct for minor differences.

FIRST CLASS MAIL