BULK RATE
U.S. POSTAGE
PAID
MADISON, WI
PERMIT 2981

COMAL Users Group
6041 Monona Drive
Madison, WI 53716

COMAL Today Issue 15 January 4, 1987

GENERAL & REFERENCE

BEGINNERS

FUN

PROGRAMMING

GRAPHICS

2.0 PACKAGES & ADVANCED

APPLICATIONS

ADVERTISERS

PUBLISHER
COMAL Users Group, U.S.A., Limited
6041 Monona Drive
Madison, WI 53716

EDITOR
Len Lindsay

ASSISTANTS
Richard Bain
Maria Lindsay
David Stidolph
Geoffrey Turney

CONTRIBUTORS
James Adams
Richard Aurland
Phyrne Bacon
Richard Bain
Norbert Bakker
Jack Baldrige
Bill C
Keith C
Borge Christensen
Captain COMAL
Linda D
Mike D
Robert D
Herbert Denaci
Doug Drake
Mike Erskine
FJ Fornorn
Brian Grainger
Bob Hoerter
Bill Inhelder
Dick Klingens

CONTRIBUTORS
Jesse Knight
Len Lindsay
John McCoy
Richard Olivieri
D Bruce Powell
Kevin Quiggle
Mike R
Joel Rea
Terry Ricketts
David S
Garold S
Sid Seiferlein
Richard Shagott
Todd Shagott
Paul Stanko
David Stidolph
Jim Ventola
Joe Visser
David Warman
Lowell Zabel

From the Editor's Disk

by Len Lindsay

We are continuing to use small pictures to identify which version of COMAL applies to each article. The **disk** means 0.14, **cartridge** means 2.0, I means IBM PC COMAL, and the **chip** means 2.0 with Super Chip.

Our experiment with the cover of *COMAL Today - The Index* worked out well. It is printed on Silver Currency stock! This issue, we are trying a special type of cover just for the holidays.

Our VOTE card insert last issue was very helpful to us. Results of cards returned are printed on page 8. It was interesting to find that 1/3 of our readers want more listings, 1/3 less, and 1/3 the same.

Now, we'd like to find out what style of listings you like. Using the smaller type listings permits not only more lines per column, but often 3 columns per page. The outlined format clearly points out the structures. We know which kind we like (no hints here). Now we want to find out your preferences. We will even give you $1 for returning the card with any order (if you order by phone, fill out your card first then be prepared to read off your answers. You still will get the $1 off).

Plus, just in time for gift giving, we are introducing a custom molded plastic case that holds up to 5 disks and 2 of our small books. Give your gift a professional look. The case then can be used for storing the items, or carrying them around (it snaps shut, dust proof inside). We are even giving these cases away FREE with several of our small book/ disk sets! Other special limited time offers are included on the back of the VOTE card inserted after page 16. Check them out.

Meanwhile, the sad news is that the declining value of the US Dollar in Denmark has eroded our purchasing power. The result is higher prices on our imported items: 2.0 cartridge, Cartridge Tutorial Binder, and IBM PC COMAL.

The good news is that by the time you read this, we should have Super Chip on Disk ready! Yes, now you can share your Super Chip programs with any cartridge owner! And, yes, now the original beige 2.0 cartridge owners can use Super Chip. More details are in the article on page 74.

We have some good programs and articles waiting in the wings. Two more Data Bases (thanks to Russell Jensen and Bill Inhelder), Fourier, a Sort package, a 2.0 Graphics Editor / Merger, Disk Manager and more. Plus, we are still anxiously waiting for the report on MacIntosh COMAL.

Now that QLink (the national Commodore On-Line Network) has improved their system to allow saving messages to disk, our COMAL section should be quite useful to you. The system is explained further on the next page. And to show you what to expect, we are printing selected messages off our Message Base. Thirteen pages full! To do this we cut out our Letters and trimmed down our Questions & Answers pages.

Those who like listings rejoice! We are printing the complete listings for 64 procedures and functions! Each also includes a small description. They are preceded by David Stidolphs introduction and MERGER system, and followed by an example of how to construct a program.

Finally, congratulations to Dick Klingens. His Package Maker article was a favorite last issue. And this issue he has three more good articles. Can he do it again? ■

COMALites Unite!

If you'd like to try out the *new* QLink online network, we have arranged for them to send you a free startup disk. See their ad on page 21. **Then to get to our section:**

Goto Commodore Information Network (CIN)
Choose Magazine Rack
Choose COMAL Today

See page 9 for examples from our Message Base. We also use it to post new information about COMAL. You can *capture* anything in the Message Base to a disk file. Taking advantage of this feature, procedure and function listings are now posted as messages. You can read them, then press the **F3** key to *capture* it to disk. Later run the following program to convert it into a mergeable file:

```
    DIM text$ OF 100, infile$ OF 16
    DIM indrive$ OF 2, outdrive$ OF 2
    startline:=9000; increment:=1
    indrive$:="0:"; outdrive$:="0:"
    linenumber:=startline
    INPUT "original filename:": infile$
    PRINT "qlink message capture file?"
    INPUT "(y or n): y"+CHR$(157): text$
    OPEN FILE 2,indrive$+infile$,READ
    OPEN FILE 3,outdrive$+"l."+infile$,WRITE
    IF text$ IN "Yy" THEN skipline(2)
==> WHILE NOT EOF(2) DO
!       INPUT FILE 2: text$
!       remove'linenumber(text$)
!       PRINT FILE 3: linenumber,text$
!       PRINT linenumber;text$
!       linenumber:+increment
--> ENDWHILE
    CLOSE
    //
==> PROC skipline(num)
!       FOR x:=1 TO num DO INPUT FILE 2: text$
--> ENDPROC skipline
    //
==> PROC remove'linenumber(REF text$) CLOSED
!       IF LEN(text$)=0 THEN text$:=" "
!   ==> IF text$(1:1) IN " 0123456789" THEN
!   !   ==> IF LEN(text$)=1 THEN
!   !   !       text$:="//" // comment
!   !   +-> ELSE
!   !   !       text$:=text$(2:LEN(text$))
!   !   !       remove'linenumber(text$)
!   !   --> ENDIF
!   --> ENDIF
--> ENDPROC remove'linenumber
```

Using this program, the posted procedures do not require line numbers! The program automatically renumbers all procedures, whether they have line numbers or not!

If your *captured* file is text, you can print out the file later right from COMAL:

```
    DIM text$ OF 100, filename$ OF 18
    INPUT "name of file to print: ": filename$
    INPUT "send to printer? y"+CHR$(157): text$
    IF text$ IN "yY" THEN SELECT OUTPUT "lp:"
    OPEN FILE 2,filename$,READ
==> WHILE NOT EOF(2) DO
!       INPUT FILE 2: text$
!       PRINT text$
--> ENDWHILE
    SELECT OUTPUT "ds:"
    CLOSE
```

To take advantage of the new QLink **Message Search** we suggest prefixing message titles with procedure, function, 0.14, 2.0, or SuperChip when applicable. Example:

SUBJ: procedure: SHIFT'WAIT

We also have a **Conference Room** inside our section where we now hold national meetings. Currently the meeting is the **second Thursday of each month at 10 pm Eastern Time**. We hope to add a second meeting each month for those who can't make the Thursday meeting. Tentatively, it will be the **third Sunday of each month at 10 pm Eastern time**.

In our conference room you can *capture* the last 80 lines of text printed in the room using the **F3** key. The disk file created can be printed later using the second COMAL program listed above.

We also have upload/ download program areas. If you upload anything into it, make sure to send an EMail message to **Donna Q**. Ask her to make it "*visible*". Refer to your QLink manual for more details. ■

Questions & Answers

CASE STATEMENT

Question: While working with data entry routine for matricies, I came across a problem with the **CASE** statement. *[The example presented here is simplified to show the problem more clearly than a matrix example would.]* It appears that it will execute correctly only the first time it is called. The second time it is called, even though x is now 2 (as shown by the print statement in line 20), it executes only line 90 which shows no match, even though x=2.

Looks like the designers of COMAL did not consider the situation where the same **CASE** statement might be called repeatedly. - Bill Inhelder

Incorrect Program:

```
0010 for x:=1 to 3 do
0020   print x;
0030   case x of
0040   when x=1
0050     print "one"
0060   when x=2
0070     print "two"
0080   otherwise
0090     print "no match"
0100   endcase
0110 endfor x
```

Answer: This is an incorrect use of the WHEN part of a CASE statement. Just values to be compared belong after the WHEN. Thus, WHEN 2 gives the value 2 to be checked for a match, while WHEN X=2 first had to analyze X=2 to be TRUE, and then used TRUE (a value of 1) as the value to be checked for a match, and 1 does not equal 2. To correct the program we need only change the two WHEN lines:

Corrected Program lines:

```
0040   when 1
0060   when 2
```

KEYBOARD BUFFER

Question: I have a question concerning 0.14 Dynamic New and Call the Next Program in *COMAL Today #13* on page 15. Would you explain what POKE 631,19 means? I know that 631 is the start of the keyboard buffer, but what is the 19 for? Or the 13 or 17? Next, what does POKE 198,3 do? - Sid Seiferlein, Muskegon, MI

Answer: As shown in our COMAL memory maps in COMAL Today #6, the keyboard buffer is in locations 631-640 and location 198 is the keystroke counter. Here is what is happening. Normally, when you press a key, the ORD value of the key is placed into the first open spot in the keyboard buffer (631) and the count of waiting keys is incremented by 1. This allows you to type some keys even while the computer is busy doing a calculation. For example, type in this short program:

```
0010 print "starting"
0020 for x=1 to 9999 do null
0030 print "ending"
```

Now, run the program. The word starting appears on the screen. Then there is a short delay. Finally, the word ending appears, followed by end at line 0030.

To see how the keyboard buffer works, run the program again, but right after the word starting appears, type a couple letters like: asd. Notice, while you type them, nothing appears on the screen! But as soon as the program ends, the word ending appears, followed by end at line 0030, just as before. But now, in addition, the letters that you typed also appear on the screen after that! This shows the keyboard buffer in action. It remembered what you typed.

More ►

This is what happens if you typed: asd. The first letter was "a". It's ord value is 65. So the number 65 is placed in the next available spot in the keyboard buffer, which is 631 since it is the first key pressed. There is now one valid key in the buffer, so the number 1 is placed into location 198, the keystroke counter.

Next is the letter "s". It's ord value is 83. So the number 83 is put into location 632, the next spot in the keyboard buffer. Now there are two valid keys in the buffer, so the number 2 is placed in location 198, the keystroke counter.

Finally, the letter "d" has an ord value of 68. Thus a 68 is placed into location 633, the next spot in the keyboard buffer. The valid key count is now three, so the number 3 is placed into location 198.

This is how the keyboard buffer works under normal circumstances. However, we can trick the computer into thinking that a key has been pressed by POKEing the proper values in the keyboard buffer and setting the keystroke counter correctly. Thus the 3 POKEd into 198 tells the computer that 3 key values in the keyboard buffer are valid. POKEing into 631-640 the ORD values of keys, provides the correct key information. Some special key values are: 19 for HOME, 147 for CLEAR, 13 for the <return> key and 17 for Cursor Down. See also procedures fillkeys and clearkeys on page 41.

FOR THE ADVENTUROUS:

Below is a short but interesting program. It works unchanged in both COMAL 0.14 and 2.0. It continually displays the contents of the keyboard buffer and the keystroke counter. To make it even more useful we added two more features:

* *Valid keys in the buffer are shown in reverse field.*
* *An arrow points at the last valid key in the buffer.*

```
print chr$(147),"hit <control> to end"
repeat
  print chr$(19) //home cursor
  print "keyboard buffer contents:"
  for x:=631 to 640 do
    if x<=peek(198)+630 then print chr$(18),
    print using "### --> ###": x,peek(x);
    print chr$(peek(x)),chr$(146);//rvs off
    if x=peek(198)+630 then
      print "<-- last key in buffer"
    else
      print tab(39) //erase to end
    endif
  endfor x
  print "buffer counter is at:";peek(198)
  print chr$(18),"valid keys are reversed"
until peek(653)=4 //control key
```

To stop the program press the <control> key - or hit the STOP key. You will note that the STOP key has a different effect on the buffer. Try running the program, typing: asd, and then hit the STOP key. Now, clear the screen, and RUN the program again. Notice that there is "junk" left in the buffer, but that the counter correctly is set at 0. Thus COMAL doesn't care what the values are, since it is not going to use any of them.

NOTES: special keys (such as <return> or cursor down) may ruin the display. It would be possible to check for the special keys, and not print them, but I wanted the example program to be short. Finally, notice that COMAL 2.0 sets the keystroke counter to 0 when the program ends, but 0.14 does not. Have fun playing with the program. ■

COMAL Clinic

VAL REVISITED (AGAIN)

On page 25 of *COMAL Today #12* we listed a val function for COMAL 0.14 (also on page 49 in this issue). We failed to mention that it would give an error if you attempt to get the numeric value of a string that does not yield a number. Len found this out the hard way when he used it in his *Doctor Who Data Base System* program.

An error is the proper response when asking for the numeric value of "ABC". "ABC" is not a number! However, a running COMAL 0.14 program cannot handle an error, so it may be more appropriate to return 0 as the value of any non numeric string. Len adapted the recursive **value** function that Borge Christensen provided us years ago. It is for positive integers only and is not fool proof. But a few checks were added to **RETURN** 0 on obvious non numeric strings. Here it is:

```
func value(s$) closed
  length=len(s$)
  if length<1 or length>5 then return 0
  ones=ord(s$(length))-ord("0")
  if ones<0 or ones>9 then return 0
  if length=1 then
    return ones
  else
    return ones+value(s$(1:length-1))*10
  endif
endfunc value
```

Further Reference:

VAL function, COMAL Today #15, page 49
VAL and STR$ in 0.14, COMAL Today #12, page 25
VAL and STR$, COMAL Today #3, page 28
STR$ and VAL Routines, COMAL Today #2, page 6
Floating Point VAL Function, COMAL Today #1, page 20

DUALING FUNCTION KEYS 2.0

One feature of the COMAL 2.0 cartridge can leave beginners bewildered. To see what I mean try this in direct mode:

1) Turn the computer off, then on.
2) Press the F3 key.
   You now are in Splitscreen mode.
3) Press the F1 key.
   You now are back on the textscreen.
4) Type: **USE system**
5) Type: **defkey(3,"test")**
   This sets F3 to print test.
6) Press the F3 key.
   Ooops ... it didn't print test.
7) Press the F1 key.
   Now we are back on the textscreen.

This demonstrates how the F1, F3, and F5 keys can alternate between your user defined function and the graphic/ textscreen switches. You can force them to change for you like this:

8) Press <*control*> and "u" together.
   This toggles the F1, F3, F5 keys.
9) Press the F3 key.
   The word test is printed.

So, <*control*>-U can be used to flip back and forth between F1, F3, and F5 definitions. However, there is a problem during a running program: <*control*>-U does not work. So, here is a procedure you can use anytime to restore the function key definitions to your user defined values:

```
PROC restore'fkeys(flag) CLOSED
  POKE 49763,(not flag)
ENDPROC restore'fkeys
```

Now, to restore the function keys to your values, just issue the command:

```
restore'fkeys(true)
```

More ►

SELECT "LP:" & "DS:"

Some operating systems can simultaneously direct output of a program to both the screen and the printer. This can be done in COMAL by using procedures:

```
PROC pr'num(x)
  SELECT "lp:"
  PRINT x
  SELECT "ds:"
  PRINT x
ENDPROC pr'num

PROC pr'str(x$)
  SELECT "lp:"
  PRINT x$
  SELECT "ds:"
  PRINT x
ENDPROC pr'str
```

Better yet, Dick Klingens has developed a package that echoes all printer output in a **PRINT** statement on the screen. The package is illustrated with the following program:

```
USE echo
setecho(TRUE) // echo on
SELECT OUTPUT "lp:"
FOR t:=1 TO 3 DO
  // now double print
  PRINT "This is the #",t
ENDFOR t
SELECT OUTPUT "ds:"
PRINT "Ready" // only on the screen
setecho(FALSE) // echo off
END ""
```

Pkg.echo and *demo/echo* are on *Today Disk #15*. Warning: **echo** changes the CHROUT routine and uses the RS232 buffer, making it incompatible with the **c128** package, **rabbit**, Super Chip (unless both are disabled), and a few other programs.

TURNTO

Turnto is powerful procedure submitted by Joel Rea to be used in connection with turtle graphics. It sets the turtle's heading to point to a specified coordinate, and turns the turtle image accordingly. A demo program is on *Today Disk #15* to show how this could be used for vanishing point perspective work.

```
FUNC deg(r) CLOSED
  RETURN 57.2957795*r
ENDFUNC deg
//
FUNC rad(d) CLOSED
  RETURN .017453295*d
ENDFUNC rad
//
FUNC rho(x,y) CLOSED
  IF ABS(x)<1e-08 THEN RETURN PI*(y<0)
  IF ABS(y)<1e-08 THEN RETURN PI/2*SGN(x)
  IF y<0 THEN RETURN (PI/2+ATN(-y/x))-PI*(x<0)
  RETURN ATN(x/y)
ENDFUNC rho
//
FUNC theta(x,y) CLOSED
  RETURN SQR(x*x+y*y)
ENDFUNC theta
//
PROC turnto(x,y)
  setheading(deg(rho(x-xcor,y-ycor)))
ENDPROC turnto
//
//A demonstration of PROC turnto(x,y).  This PROCedure
//is useful for vanishing-point perspective work!
//
USE turtle
graphicscreen(0); window(-1,1,-1,1); ht
FOR i:=-1 TO 1.05 STEP .1 DO
  moveto(i,-1); turnto(0,0); fd(theta(i,-1)*.75)
  moveto(i,1); turnto(0,0); fd(theta(i,1)*.75)
  moveto(-1,i); turnto(0,0); fd(theta(-1,i)*.75)
  moveto(1,i); turnto(0,0); fd(theta(1,i)*.75)
ENDFOR i
penup; home; st
REPEAT  UNTIL KEY$>""0""
//
//COMAL 0.14 Notes:  The FUNCs and PROC turnto all work
//in 0.14 as is. The demonstration does NOT!
```

Vote

Last issue we included a "Vote Postcard" between pages 16 & 17. We are doing the same thing this issue. But we are giving you $1 off any order if you return it with an order (or give us the replies over the phone - fill out the card before you call so you are ready with the answers). Of course, you can mail it by itself too. Just fill it out, put it in an envelope, and mail it. Be counted! Mail it today! We need it by January 10, 1987 at the latest!

Here are the results from Issue #14:

Favorite Article
#1 - Program Outliner, Len Lindsay
tie - Package Maker, Dick Klingens
#2 - Scope, Richard Bain
#3 - Listerine, Will Bow
tie - Multi Function Graphics, Lowell Toms

Favorite Authors
Len Lindsay, David Stidolph, Richard Bain, Kevin Quiggle, and Colin Thompson

Favorite Book
#1 - COMAL Handbook
#2 - Cartridge Tutorial Binder

Favorite Disk
#1 - Today Disk #11
#2 - COMAL 2.0 Packages Disk

Favorite Program
#1 - Pop Over System

Program Listings
35% - More Listings
35% - Keep the Same
30% - Less Listings

Version of COMAL Used (some overlap)
82% - COMAL 2.0
52% - COMAL 0.14
30% - COMAL 2.0 with Super Chip
9% - IBM PC COMAL 2.0 ■

String Bug

```
0010 dim text$ of 10
0020 text$:="happy"
0030 testing
0040 print text$
0050 //
0060 proc testing closed
0070  dim text$ of 10
0080  text$:="sad"
0090  // text$(1:3):="sad"
0100  print text$
0120 endproc testing
```

When this program is **RUN** from COMAL 0.14, it correctly prints the words *sad* and *happy. Sad* is first printed from procedure **testing**. Since the procedure is **CLOSED**, setting a new value for **text$** inside it does not change the value of **text$** outside it. Therefore, *happy* is printed below *sad.*

Now, type **DEL 70** and **RUN** the program again. In line 100, you will get a *string not dimensioned* error. This is true, but why didn't we get this error in line 80? Type **DEL 100** and **RUN** the program a third time. This time, *sad* is printed outside the procedure. This is clearly incorrect. A **CLOSED** procedure can't change the value of a variable outside it, can it? Well, it isn't susposed to, but it did.

Now type **DEL 80** and remove the comments from line 90. **RUN** the program once more to get the proper error message in line 90.

It seems you can assign a string to a global string variable (but not a substring) from within a **CLOSED** procedure. However, you can't access the variable to **PRINT** it. As it turns out, you can't assign the value of the global string variable to another string within the **CLOSED** procedure either. There is no problem with reals, integers, or arrays of any type. COMAL 2.0 doesn't have this problem at all. ■

QLink - The Message Base

PUNTER PROTOCOL

SUBJ: punter protocol package (R1)
FROM: C64UGOSJ 09/17/86 S#: 9174

I have downloaded and assembled source code of the new punter protocol. My question is:
Is there any PUNTER PACKAGE available?
Would anyone be interested in bettering the library of Comal 2.0 users by adding this powerful program in a LINKed package? I have tried unsuccessfully!
Please help! C64 UGOSJ

SUBJ: Any takers?
FROM: Captain C 09/25/86 S#: 29876

This seems to be a worthwhile project.

There are several MODEM TERMINAL programs for COMAL 2.0 on our MODEM disk - none include Punter Protocol. We also have a COMAL 2.0 BBS system on our Shareware Disk #1.

We also may be able to assist you from our COMAL offices here if you would send your files on disk to us.

Depending on what memory locations it requires and if it can be relocated if in conflict, we may be able to to get it into a proper COMAL 2.0 PACKAGE - then it can link onto our MODEM programs as well as the BBS.

Thanks ... Len Lindsay

COMAL IN FORTH

SUBJ: Needed -- FORTH programmers!
FROM: COMALite J 09/01/86 S#: 53072

Since COMAL programs compile into a sort of Forth-like code, I figured that it should be possible to write a COMAL in Forth! I think this project would be useful for the following reasons:

* Since at least one version of Forth exists on just about every CPU, computer system and operating system in all existence, from the likes of the Timex/ Sinclair 1000 and CBM VIC-20 to the CBM Amiga and even large mainframes, such a COMAL would be easily portable to a LOT of computers!! Note: At present, this is the best possible hope for an Apple COMAL!

* The Forth language itself is Public- Domain! In fact, there is even a P-D Forth for the C-64/C-128 available free for the downloading, right here on QL!! It is called "Blazin' Forth", and it is an enhanced implementation of Forth-83, the latest standard. I have already downloaded it. Even the SOURCE is on- line! It is in the Software Libraries, in the Programming section under Other Languages. Anyway, the COMAL we would write would ALSO be Public Domain (or at least Shareware) to provide the widest possible distribution and eventual acceptance of COMAL!

* Forth executes up to 75% the speed of pure hand-coded machine language (1 of the reasons COMAL is so fast?) and is usually even MORE COMPACT than pure machine-language! Also, parts of a Forth program can be re-coded into assembly (included) for maximum speed! Thus, we can add more features while keeping a larger workspace for the user than, say, COMAL 0.14!

* Forth supports true 16 and 32 bit signed and unsigned integer math! Both COMALs for the C-64 allow integer variables, but only perform Real (floating- point) math (COMAL 2.0 does do true integer math during indexing, either with FOR loops or array/string subscripts). This slows them down. We could provide true integer math in our "COMAL-F" to offset the slight speed degradation of Forth.

Anyway, I am just learning Forth. I would like the aid of any experienced Forthers I can find! I think this is a very important & feasible project! If anyone is interested in helping out, leave a reply here and/or notify me via E-Mail. Thanks!

TEXT WINDOWS

SUBJ: windows on textscreen (2.0)(R3)
FROM: LindaD 08/17/86 S#: 32349

Is there anyway to use windows on the textscreen? I would like to be able to superimpose help menus or the like over whatever screen is currently displayed on the textscreen. Also along that line, how can I print a corner (as in a menu box) on the textscreen? A straigt line graphics char. is available but I can't get corners to print.

Anybody have any ideas or suggestion?

Thanks,
linda

SUBJ: try pop ups (R)
FROM: Terry R 08/21/86 S#: 51117

Comal Today #11 has a great article on page 18 showing how to have menus pop onto the text screen by using the 'run/stop' key. Check it out. I think it will provide much of what you want.

SUBJ: corners (R)
FROM: Captain C 08/21/86 S#: 51499

Corners are made with graphic symbols obtained with the SHIFT or the COMMODORE key.

Example: try this:

```
dim text$ of 1
print chr$(142);chr$(147)
repeat
  text$=key$
  if text$>"" then print text$,
until true=false // forever.
```

Now RUN that program.

Try holding down the COMMODORE key (under the STOP key) - and simultaneously pressing A - then S - then Z then X. Those are the square corners.

Then try SHIFT with U, I, J, K. Those are the round corners.

Regards ... Len Lindsay

FASTLOADERS

SUBJ: magnum load & comal 2.0 (R2)
FROM: LindaD 08/17/86 S#: 32299

Has anyone been able to figure out how to use Magnum Load (a replacement Kernal Rom fastloader) with the 2.0 cartridge. The sys and pokes given to activate it don't work. 1 sends the computer to never never land and the other doesn't seem to do anything. The other option of holding the F1 key on booting up only gives me the Comal Renum option. Has anyone had

More ►

any luck with this or any ideas?
linda

SUBJ: use of ram (R)
FROM: Terry R 08/21/86 S#: 51196

I am not familiar with this particular fast loader, but most require that the program be loaded into ram. This will not work with Comal 2.0, since it uses ALL of ram (except a small portion of field C).

I suspect that this program tries to load into the ram under the kernal rom. That is the location that comal uses for its graphics screen. Your best bet when using Comal is to get the new Super Chip which includes a fast loader, & not try to use any other fast loader software.

SUBJ: fast load
FROM: Captain C 08/21/86 S#: 51547

Terry is correct. General purpose fastloaders will not work with COMAL 2.0.

We have a chip called SUPER CHIP that plugs into the empty socket in the black COMAL 2.0 cartridge - it includes the RABBIT fastloader. This fastloader is also available disk loaded on the TODAY DISK #12.

To enable it:

From chip:

```
USE RABBIT
SETFAST(TRUE)
```

From disk:

```
LINK "PKG.RABBIT"
USE RABBIT
SETFAST(TRUE)
```

Both are available from COMAL Users Group USA.

Regards ... Len Lindsay

QLINK COMAL

SUBJ: COMAL in Hiding (R3)
FROM: Mr Bill C 09/21/86 S#: 20125

I've been a COMAL enthusiast for quite a while now, but a QLink user for just over a month. I looked for you folks off and on during that month. Finally, a small reference in COMAL Today 13 rang a bell. I looked in the CIN Magazine Rack. Why is COMAL hiding in the magazine rack? THIS IS AN IMPORTANT LANGUAGE, FOLKS! Capt. COMAL, can't you get this group installed more prominently where others can find it easily?

C O M A L P O W E R !

-Bill Calkins

SUBJ: Hiding (R)
FROM: KeithC4 09/24/86 S#: 29146

I agree. Where is everyone? I'd like to let QLink know how many COMALites are here and serious about COMAL programming!

Just hope everyone can find this section.

KeithC4

SUBJ: This location (R)
FROM: Captain C 09/24/86 S#: 29821

This location for COMAL is not of our own choosing. We had to take what QLINK offered. While I agree that the Magazine Rack is not a likely place to look for COMAL ... we are here and offer answers to questions posted here. You also are welcome to upload your COMAL programs in the adjoining area. When you do, please send an EMAIL note to:

DONNA Q

Advise her that you uploaded a program to the COMAL section and wish it to be made "visible".

Thanks ... Len Lindsay

SUBJ: COMAL on QLINK (R)
FROM: Monitor 09/25/86 S#: 31353

Why can't the COMAL op system be put in the QLink download libraries? It would be faster to distribute and could possibly become a BIG(GER) and (more) popular language. I can't get a copy anywhere in Nashville (TN). I am currently fluent in BASIC and halfway through Pascal. Now I'm interested in COMAL. Please help.
TM

SUBJ: COMAL System on QL
FROM: Captain C 09/25/86 S#: 32064

At this time we are not including the COMAL 0.14 system itself in our download section on QLink for several reasons.

* COMAL 0.14 is copyrighted. We include written permission to copy it when we send out our disks. Placing it on a BBS system could invalidate our copyright.

* COMAL 0.14 system consists of several interlinked files - totalling about 150 blocks. It is not just one file. You need the HI, BOOT, and ERROR files for a good working COMAL system.

* Download time would be extensive due to the large file size.

* COMAL 0.14 system is available on disk at a VERY reasonable price - including many demo / useful programs.

* COMAL 0.14 is available from most local Commodore User Groups.

If you can't find COMAL 0.14 - just send $9.75 plus $2 shipping to:

COMAL Users Group USA
6041 Monona Drive
Madison, WI 53716

Ask for the QLink COMAL special disk.

Regards ... Len Lindsay

DISK TO SCREEN/PRINTER

SUBJ: HELP: 2.0 Spool SEQ to DS/L(R3)
FROM: Garold S 08/04/86 S#: 33158

I am trying to find a way to spool a SEQ file off my disk to the screen and alternatively to the printer, using one or two simple lines of keyboard commands with my COMAL 2.0 cartridge. I am trying to find a way to read my Q-Mail off the disk I save it to, without loading a reader program like Sprint.

It seems that COMAL 2.0's powerful I/O redirection would make that possible.

Here is what I've tried so far:

```
Select Input "filename"
```

Amazingly, this spits the seq file text onto the screen. Unfortunately, COMAL wants to give me its opinion of whether each line of text is

More ►

syntactically correct, and the result is a mess of lines that overwrite each other. Try it.

Can anyone suggest a procedure to SPOOL a SEQ disk file to screen or printer?

Can anyone tell me how to make COMAL stay silent during my Select Input procedure?

Any assistance appreciated.

SUBJ: COMAL, no. SuperChip, si (R)
FROM: COMALite J 08/06/86 S#: 40780

This can't easily be done in COMAL without a small PROCedure, but it is a CINCH with COMAL 2.0 and the new SuperChip! Here's how:

```
   USE files
   SELECT OUTPUT "lp:"
* To redirect output to, say, a
printer.
   type("filename")
* May be ANY non- REL file, even a
non-DISK file!
   SELECT OUTPUT "ds:"
* If the output has been redirected.
```

Without SuperChip, you can make your own "type" PROCedure (one is included in my COMAL Machine-Language Monitors (CMLM's)) like this:

```
   PROC type(filename$)
    OPEN FILE 8,filename$,read
    REPEAT PRINT get$(8,254), UNTIL
    EOF (8) // Wrap Line
    CLOSE FILE 8
   ENDPROC type
```

It IS also possible to do a true "spool" of an ASCII or "PetASCII" non RELative file to the printer (but NOT the screen). I have done it with a VERY short ML routine from BASIC, and I see no reason why it should not work as a COMAL package! It doesn't even use the C-64's interrupts, or the C-64 at ALL for that matter (once the "spool" has started)! With that, you could type, say:

```
   LINK"obj.spool"
* One time only!
   USE spooler
* One time only!
   spool("filename")
* Must be a DISK-
   based ASCII or
   "PetASCII" non
   REL file.
```

At this point, "spool" would OPEN the file for READ and "lp:" for WRITE, then "fool" COMAL into thinking that they are CLOSEd! It would then use low level Kernal calls to tell the disk drive to TALK and the printer to LISTEN, and then remove the C-64 from the serial bus. You will have to hit RUN-STOP/RESTORE when it is done printing, but in the mean time you can use your C-64 as normal, so long as you don't use the serial bus. You can even unplug the serial bus cable from the back of the 64, and plug in another drive & use it or even TURN OFF THE C-64, and the file will KEEP PRINTING!!! I'll upload this into the COMAL 2.0 area as soon as I re-write it as a COMAL package! Watch for it!

SUBJ: Spool file (R)
FROM: FJ Fornorn 08/08/86 S#: 50805

Garold, I don't think you're gonna pull this off with mere selects. A PROC will be necessary. Here's what I did: I went back thru my COMAL 0.14 disks. I was sure I ran a SEQ file reader there. Found one - "see'roll". It shows a SEQ file while you press the shift key. I listed it from 0.14, entered it in 2.0 (along with "shift'wait", its companion), renamed all the references to it to "see". Then, do a "scan", and then a "cat". Cursor up to the file you want to look at. and type: see("filename") over the directory. As you press shift, you view the file!

Send me Email if you want any more info.

SUBJ: Printing SEQ files quickly
FROM: COMALite D 08/14/86 S#: 16579

A simple and quick way to view a SEQ file is to type the following 2 lines in immediate, or command mode:

```
OPEN 1,"filename",READ
WHILE NOT EOF(1) DO PRINT GET$(1,1),
```

When the file is finished printing, or you press the STOP key, you must close the file with:

```
CLOSE    or    CLOSE 1
```

If you want the file to go to the printer then issue the SELECT command before using the above commands. I used this method until the TYPE command was done for Super Chip.

There is no way to stop COMAL from interpreting the lines it reads from a SELECT INPUT command (unless it is from within a running program). This command is only used for BATCH files.

I know this solution is simple, and lacks even a PAUSE control (although you can use the CTRL key to slow the scrolling), but it is useful.

[Editors Note: Dick Klingens & the Dutch COMAL Group have provided us with ECHO, a package that will output to two simultaneous locations. It should be in COMAL Today 15 and on Today Disk 15]

PROCEDURES WANTED

SUBJ: Procedures needed... (R1)
FROM: DavidS6 07/31/86 S#: 2284

I am looking for procedures/ programs that will do the following:

* convert textscreens to hi-res
* convert "News Room" art to 2.0
* "rubberband" -- by this I mean the feature common to many drawing programs where a shape (box, circle) or line is stretched out on the screen before it is set on the screen.

Any help would be appreciated for any or all of these topics: if you know of a direction I can go to possibly find out more, please let me know.

Thanks,
-- David S6

SUBJ: Textscreen to Hi-Res
FROM: Captain C 08/01/86 S#: 7414

See page 34 of COMAL TODAY #13 for a very short program to copy the text screen onto the Hi-Res screen. It is listed - only about 25 lines. This is a GOOD one too - it copies custom fonts as well as normal - it is a pixel by pixel copy.

Thanks to Patrick Roye for the routine.

More ►

[Editors Note: You now may be able to "rubberband" easily using the "flip pixel" type of plotting in COMAL 2.0. Issue the following POKE and watch what happens as you plot a shape twice:

POKE $c462,$5d // sets flip mode

To return to normal plot mode issue this command:

POKE $c462,$1d // sets normal]

ENCRYPT

SUBJ: Encryption (R2)
FROM: Icarus 07/23/86 S#: 36847

Looking at the listing for Joel Rea's encryption program in CT #13, I did not find the code for reading a file that has been encrypted. Am I missing something?

SUBJ: DeCrypt (R)
FROM: Captain C 07/23/86 S#: 38841

Reference: COMAL Today #13, page 55

See the second paragraph for info on how to decrypt a file.

To decrypt the file, you must ENCRYPT it again WITH THE SAME KEY.

I know that sounds strange, but that is how his program is set up. Just use the same key for the ENCRYPT as for the DECRYPT.

Thanks ... Len Lindsay.

SUBJ: More on ENCRYPTOR:
FROM: COMALite J 07/24/86 S#: 39254

"C2.ENCRYPT FILE" is available for download in our own COMAL 2.0 library in this section!

"C2.ENCRYPT DEMO", also in that area, is a program that lets you experiment with my encryption algorithm before committing any files to it. I believe I have converted both of those programs to my "easy-read" list format, which reads easier than the listing published in COMAL TODAY #13.

You can Encrypt in MULTIPLE PASSES for extra security. Just remember each key you use. To decrypt, you must re-encrypt using the SAME keys in the REVERSE ORDER! For example, if you Encrypt using "3.14159", "2.71828" and "555.1212", you must Decrypt using "555 .1212", "2.71828" and "3.14159" in that order!

Due to the VERY powerful Device Independent File Naming Scheme of COMAL 2.0, you can do a LOT with ENCRYPT FILE besides encrypting from one file to another!

By specifying the output to "ds:" or "lp:", you can either see the results of an encryption before committing it to a file, or you can view or print a de-crypted version of an encrypted file without leaving a de-crypted copy around for snoops!

You can also (assuming you already have connection) use ENCRYPT FILE to send a file over the modem encrypted and have it de-crypted and automatically saved on the other end! All you have to do is have a copy of ENCRYPT FILE on both ends, and have an agreed- upon key. The sender sets OUTPUT to "sp:" (with any needed parms), and the receiver sets INPUT to "sp:". You send an ordinary file and he gets an ordinary file, but if anyone were tapping into the line, all they would get would be gibberish!

C128 PACKAGE

SUBJ: pkg.c128 (R3)
FROM: Prof 07/22/86 S#: 36555

Is anybody else having problem getting the pkg.to operate? I linked the pkg. However I don't get the cursor back. I checked the 80 col screen and all I have are a lot of verticle different colored lines. Also I loaded the vdc prg and it does the same thing.

Need some input on this thanks.

SUBJ: pkg.c128 -- what's wrong? (R)
FROM: COMALite D 07/24/86 S#: 40794

I don't understand the problem you are having with the c128 package. I did find a problem yesterday with certain C128's (the new ones at Sears). The 80 column screen seems to roll, but it does not scramble the screen, or crash the computer. I would like to get details on the equipment you are using (what is your power supply rated at), and what kind of monitor are you using.

I am uploading the new c128 package and vdc-editor.pop program. Please try them and EMAIL me your results.

Remember, these programs are strictly for the C128 and COMAL 2.0 users (black cartridge).

SUBJ: pkg.c128 (R)
FROM: KRM 07/31/86 S#: 63792

Will the c128 pkg work with the tan 2.0 cartridge? I have found that the tan cartridge seems to work fine on my C128.

SUBJ: it should
FROM: Captain C 09/25/86 S#: 32129

If your beige 2.0 cart works on your C128 - then it should handle our disk loaded C128 package.

Regards ... Len Lindsay

FREQUENCY / PLAYSCORE

SUBJ: FREQUENCY and PLAYSCORE (R2)
FROM: Robert D19 07/15/86 S#: 21617

I was reading COMAL Today issue #11 and I noticed someone mentioning that FREQUENCY and PLAYSCORE are not mentioned in the graphics handbook. What are these functions and how do you use them?

SUBJ: PLAYSCORE and FREQUENCY (R)
FROM: COMALite J 07/16/86 S#: 22163

In the "Cartridge Graphics and Sound and Other Packages" book by Captain COMAL), a couple of accidental omissions occurred:

On pages 52 and 53, the Automatic Sound Control for Interrupt Driven Sound (or Music) is described. The two PROCedures and one FUNCtion described there provide:

1. A way to set up arrays of note frequencies and lengths.
   PROC setscore(int#, REF int#(), REF int#(), REF int#())

2. A way to stop zero or more voices.
   PROC stopplay(int#,int#,int#)

3. A way to test if any particular voices are still playing.

More ►

```
    FUNC waitscore(int#,int#,int#)
```

What is missing? Of coure! A way to START the music! That is what PLAYSCORE is for. It is a PROCedure:

```
   PROC playscore(int#,int#,int#)
```

The three arguments represent the three SID voices. They must have been previously set up with SETSCORE. For each voice whose corresponding argument is non-zero (TRUE), that voice starts PLAYing its SCORE! It's that simple! It keeps going, without pausing your COMAL program, until the scores run out or STOPPLAY is executed on all playing voices!

FREQUENCY is a little more esoteric. On pages 44 and 45 the PROCedures SETFREQUENCY and NOTE are described. They are as follows:

1. Set voice to a numeric frequency.
   PROC setfrequency(int#, real)

2. Set voice to a frequency computed from a note value specified in a string (such as "C4" or "F7#").
   PROC note(int#, string$)

Well, that allows you to set a frequency either by absolute value or by a note. What more do you need? There is a "middleman" of sorts:

```
    FUNC frequency(<string>)
```

Frequency takes a note-string (same as the NOTE procedure), and RETURNS the ACTUAL FREQUENCY! Neat, huh? That means that:

```
   note(1,"c4")
```

is equivilant to:

```
   setfrequency(1,frequency("c4"))
```

Why have FREQUENCY then? Well, for most musical purposes it isn't necessary. But, if you are interested in absolute harmonics or just temprement, where chords are mathematically exact relations to the fundamental, FREQUENCY lets you do this to even-tempered (normal) notes! For example, try

```
USE sound
//
PAGE
PRINT AT 10,5:"Playing even-tempered"
PRINT "C-major chord:"
PRINT
PRINT "Press space bar to change:"
//
FOR voice#:=1 TO 3
 soundtype(voice#, 2)
 adsr(voice#, 5, 5, 10, 5)
ENDFOR i#
//
note(1,"c3")
volume(15)
//
LOOP
 PRINT AT 10,13: "even"
 even'tempered'chord
 REPEAT  UNTIL key$=" "
 PRINT AT 10,13: "just"
 just'tempered'chord
 REPEAT  UNTIL key$=" "
ENDLOOP
//
PROC even'tempered'chord
 FOR v#:=1 TO 3 DO gate(v#,FALSE)
 note(2,"e3")
 note(3,"g3")
 FOR v#:=1 TO 3 DO gate(v#,TRUE)
ENDPROC even'tempered'chord
//
PROC just'tempered'chord
 FOR v#:=1 TO 3 DO gate(v#,FALSE)
 setfrequency(2,frequency("c3")*1.25)
 setfrequency(3,frequency("c3")*1.5)
 FOR v#:=1 TO 3 DO gate(v#,TRUE)
ENDPROC just'tempered'chord
```

This program uses the FUNCtion "FREQUENCY" to demonstrate the difference in the even-tempered and just-tempered scales. Note that while most Western (not Country & Western, but the Western Civilization!) music uses the even-tempered scale invented by Johann Sebastian Bach, Barbershop and Sweet Adeline quartets and choruses use the just-tempered scale. That's why their harmony sounds so "close" -- and why they are NEVER accompanied, but are ALWAYS acapello! Musical instruments are tuned to the EVEN-tempered scale! The two would clash -- HORRIBLY!

Another example of the use of FREQUENCY would be in generating precise, musical effects with RINGMOD! As you may know, Ring Modulation requires precise relationships between the two frequencies involved to produce particular effects. SYNC is similar!

I recently used FREQUENCY to simulate Westminster Chimes. If anyone is interested, I'll upload it.

SUBJ: MUSIC
FROM: Captain C 07/16/86 S#: 23245

Thank You Joel - wonderful response.

Also, check COMAL TODAY #12 for the complete summary of all commands in the SOUND package.

RANDOM FILES

SUBJ: Random File Disk Bug (?) (R1)
FROM: MikeD5 07/13/86 S#: 15546

I've been continuing work on my long COMAL program, and I have almost reached completion. However, there is a bug with my random file procedure, and I have narrowed down the error to be occurring within the procedure. The procedure reads as follows:

```
PROC ins(a$,b$,c,d$,e,f$)
 OPEN FILE 4,"auto-log.dat",RANDOM 70
 WRITE FILE 4,rec#: a$,b$,c,d$,e,f$
 CLOSE FILE 4
 rec#:+1
ENDPROC ins
```

The declaration of RANDOM 70 is correct, so there is no problem with length. The routine seems to work about 90% of the time, however, once in a while when I examine the disk contents I receive an incorrect value.

I heard something about there is in existance a bug in the random file routines in Comal 0.14. Could this be afflicting my program, and if so, how do I correct it?

MikeD5

SUBJ: RANDOM BUG IN 1541
FROM: Captain C 07/14/86 S#: 19520

The 1541 has a bug in dealing with random files. COMAL goes too fast and gets ahead of the drive and the drive gets mixed up.

First - make sure the random file is the ONLY file open! Next - after your WRITE FILE statement add one line - a READ FILE statement that simply reads back the values just written --- then CLOSE the file.

More ►

Also --- I don't see why you are incrementing the REC# variable inside the read routine. It should be done in your main routine.

That should do it.

Regards ... Len Lindsay

STACK OVERFLOW

SUBJ: STACK ERROR (Error #11) (R3)
FROM: MikeD5 06/21/86 S#: 21488

Well, I finished one of my prized programs, with 348 bytes to spare. The program worked well until I made some more additions, and now I keep getting a STACK ERROR (Error #11). This error message just keeps scrolling down the screen.

Can someone give me all possible conditions under which this error can occur? I would appreciate it very greatly!

MikeD5

SUBJ: stack overflow (R)
FROM: Captain C 06/23/86 S#: 27874

COMAL TODAY issue 2, page 15 has a good start at what you need. It lists 7 ways to create a STACK OVERFLOW ERROR.

Since you have a large program, your problem is probably running out of memory to use for the STACK. As the program runs, the stack is dynamic, growing as needed to keep track of procedure calls, loops, etc. CLOSED procedures and functions are especially memory hoggish!

Solutions:

First, clean up your name table! To do this, LOAD your program. Do NOT run it yet! Now, list it to disk:

LIST "TEMP.LST"

Then erase it:

NEW

Now, enter it back:

ENTER "TEMP.LST"

Finally - before you run it, save it again with a new name:

SAVE "NEWTRY"

After that, try running it. If it still gives STACK OVERFLOW, delete all the remarks in the program. Next, see if some CLOSED procedures can be made open (delete the CLOSED from the header).

Between all this, you should be clear of the STACK OVERFLOW.

To see the other posibilities, check out COMAL TODAY #2 (part of the COMAL YESTERDAY book now).

Regards ... Len Lindsay

SUBJ: Expanding Memory (R)
FROM: COMALite D 06/25/86 S#: 34271

COMAL 0.14 normally provides 9902 bytes free on power up. A procedure was printed back in COMAL Today #5 that expanded memory space to 11,838 bytes free. If you have any recent (within the last year or so) disk it should expand memory for you. Make sure you issue a NEW to enable the new memory.

If you have a large program in memory, issue the NEW command before LOADing in the next program. COMAL seems to have a bug in the LOAD and CHAIN commands that don't fully free memory between programs.

Also, INTEGER variables (names end with a # - as in "loop#") save 3 bytes over real variables. This can save lots of space in arrays.

SUBJ: Thanks
FROM: MikeD5 07/05/86 S#: 61039

Thanks Captain C and Comalite D for your help... I'm going to try out your suggestions this afternoon and then I'll get back to you.

I think it is the memory problem at best... The program has approximately 300-400 bytes left when running.

MODEM CONTROL IN 2.0

SUBJ: COMAL modem: CT#9 (R3)
FROM: Icarus 06/18/86 S#: 14512

In COMAL TODAY #9 there is an article about COMAL controlling the modem directly. A friend and I just spent two hours on the phone trying to figure out what the author was not telling us. I assume the system can work, but there are no step by step instructions for how two COMALites with the cart and a phone connection can send LISTings to each other.

Can anyone help? Specifically, how do I send the high pitched whine to my friend? Who should be in originate mode? Etc. Also, I notice from the manual that select "sp:" sets up some defaults, and these are not the defaults usually used in C64 to C64 setups. Could this be the problem?

Thanks for any tips.

SUBJ: Hope this helps . . . (R)
FROM: COMALite J 06/19/86 S#: 15757

Try using "sp:/a-" on both ends, since ASCII conversion is obviously not needed on C64-C64 transfers. Generally, the caller should be set to "Originate", and the callee to "Answer", but all that really matters is that they not both be the same! When activated, an "Answer" modem sends the high tone, and when the "Originate" modem "hears" it, it responds with a lower tone. The means of activation differ between modems. Read your instruction manual and look for POKEs to do "off-hook" to activate, and "on-hook" to hang up.

SUBJ: modem to modem (R)
FROM: Captain C 06/19/86 S#: 16822

As many of you may know - I basically dislike modems. I'm here to support COMAL.

So, it is a good sign that even I could have one C64 call another one and test the built in Modem stuff in the COMAL 2.0 Cartridge. Of course, having two phone lines and 2 C64's with modems helps.

What you might try. Call the person you wish to do a modem transfer with - voice call. Both of you get your computers set up - turned on, modem in, cart in. One set modem to Originate - other to Answer (if no switch - then don't worry - modem will figure it out). Then both of you switch over to data transmission - you then are connected. One of you then does a:

More ►

SELECT INPUT "SP:"

The other does a:

LIST "SP:"

The program does not list to the screen - rather it goes over the modem.

Or - if one of you has an auto answering modem - call it with your modem - it will answer and no voice communications needed.

It really is fun to take over another COMAL computer by phone!

Regards ... Len Lindsay

SUBJ: Cart to Cart transfers
FROM: COMALite D 06/25/86 S#: 34333

If both you and your friend have COMAL 2.0 cartridges, try the following.

The sender loads the COMAL program into memory (with the modem hooked up and set to originate mode, but not on). Now call the other person.

Have the other person hook up their modem to the computer (in answer mode) and type the following line in COMAL:

ENTER "sp:"

The original caller now types:

LIST "sp:"

This should transfer the program without a modem program!

REVERSE CHARACTERS

SUBJ: RVS messages in editor (R3)
FROM: Icarus 05/24/86 S#: 34585

I recently had a chance to work with a COMAL 2.01 program running on an SX-64. This must have been a newish cart since a copy of the Tutorial Binder was with it. I noticed that when an error was entered the system showed the Keywords and variables in Reverse, with some sort of color coding. The program I was working on did some fancy stuff with graphics. Is there a COMAL that behaves that way? Is it due to the different ROM in the SX? Or was it the program? Just curious.

SUBJ: RVS characters (R)
FROM: COMALite R 05/25/86 S#: 36135

If the program was using the Extended color mode then the upcase characters would appear in reverse; I believe. I have a grey and black cartridge and have used both with my SX-64 without any noticeable difference in color when in the edit mode.

Hope this helps!
...Rich

p.s. Nice to see you on!

SUBJ: RVS characters (R)
FROM: Icarus 05/25/86 S#: 36678

Thanks, Rich. It DID use extended colors as I recall. The effect was nice; that's why I thought it might have been there on purpose.

SUBJ: RVS alternative
FROM: Captain C 05/27/86 S#: 39292

Extended Color mode can be used for some very interesting effects - and probably is the answer to your question. I believe David Stidolph (COMALite D) uses it in some of the HI programs to produce the "blinking" color boxes effect.

There is one other possibility I can think of - perhaps the cartridge you used had a custom EPROM installed in the empty socket. Packages can be included that automatically initialize and can INTERCEPT errors! Also can intercept the RUN command and such.
Interesting uses!!!!

Regards ... Len Lindsay

COLOR MEMORY

SUBJ: Color/Screen Memory! (R5)
FROM: MikeR1 05/21/86 S#: 31852

Okay, where DID the Color and Screen Memory's go!!!! Someone please help! I would look it up in a memory map but I don't really want to bother trying to learn how to understand the map. Besides, I heard from a friend that you cannot easily poke/peek to/from screen memory. Is this true?

SUBJ: 0.14 & 2.0 Mem Maps (R)
FROM: COMALite R 05/21/86 S#: 32007

Mike which version of COMAL? I believe screen memory for 2.0 moves around. COMAL Today issue# 6 has memory maps for 0.14 and 2.0 . According to the memory map for 0.14, text screen memory is at 1024-2023 and the start of the screen character colors & graphics screen is found at 55296 (I suspect that is a two byte address). The graphics bit map is suppose to be 57344-65535.

I'm sure COMALite J and COMALite D can add much more info.

Since getting COMAL over two years ago, I have stayed away from peeks and pokes (haven't really needed them for text printing). Peeks and pokes also make it difficult to port my C64 COMAL programs over to PC-COMAL.

Hope this helps some,

...RICH

SUBJ: Display RAM areas (R)
FROM: COMALite J 05/23/86 S#: 33098

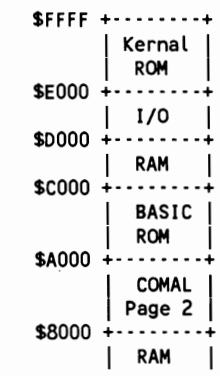
First off, the Color Nybbles, that 1k area of 4-bit (nybble) RAM that the 6567 VIC-II chip uses to hold the colors of each text cell in text and multi-res modes, is ALWAYS located starting at $D800 (55296), no matter what!

The text screen, 1k of byte RAM for holding the display codes of each character on the screen, normally starts at $0400 (1024) in BASIC 2.0 & COMAL 0.14, and also in COMAL 2.01 when not using a FONT. When using a FONT in COMAL 2.01, the screen moves to $6C00, with the font itself starting at $7000 (4k).

When using a bitmap, in either Mode 0 (hi-res) or 1 (multi-color), in either COMAL, the bitmap is stored in 8k starting at $E000 in RAM under the Kernal, and the text screen (for holding 2 colors per cell for Mode 1 and both colors per cell in Mode 0) starts at $DC00 in the RAM under the I/O.

PEEK and POKE normally see a memory map as follows:

More ►



They can be changed to view ALL 64k as RAM by executing:

```
USE system
setpage(0)
```

Other values for "setpage" can give access to CharGen ROM, other COMAL ROM pages, etc. See Jesse Knight's "COMAL 2.0 Packages" book for more info on structure of "Page" byte.

NOTE: "setpage" ONLY affects PEEK, POKE and SYS -- it doesn't instantly change the mapping, and thus won't in itself crash COMAL!

SUBJ: 0.14 Screen Memory (R)
FROM: MikeR1 05/25/86 S#: 35839

Oops! Well, what I am trying to do is make a window feature for COMAL 0.14 for my program. When I peek from 1024 and the poke to a 'free' memory location and then poke it back to mem area 1024, I get a lot of reverse spaces. Also, when I poke to color mem area it locks up! Any ideas?

SUBJ: GETCOLOR
FROM: Captain C 05/27/86 S#: 39275

In COMAL 0.14 you can use the built-in command:

```
GETCOLOR
```

It will return the color number of the pixel coordinate you specify.

```
mycolor=getcolor(50,50)
```

This will look at the pixel at the coordinate 50,50 and return the color number. If it was red - it would return 2.

This may be what you really are looking for?

Regards ... Len Lindsay

BASIC TO COMAL

SUBJ: Loading BASIC into Comal (R2)
FROM: Tarz 05/17/86 S#: 27469

I have some DATA statements from a BASIC program that I would like to LOAD into COMAL 2.0. How would I go about doing this?

Thanks in advance...TARZ

SUBJ: BASIC DATA STATEMENTS (R)
FROM: Captain C 05/17/86 S#: 28219

There may be several ways to accomplish this. I will present one way that I believe would work (with a variation at the end).

First, get the BASIC data statements stored on disk as a SEQ TEXT file. Let's say your data statements are at lines 800-999:

(remember - we are in BASIC now - and your BASIC program is currently in the computer)

```
OPEN 1,8,1,"temp,w,s"
CMD 1
LIST 800-999
PRINT#1
CLOSE1
```

That should do it. Kind of. Remember - BASIC is not as smart as COMAL. The file we just created has 4 junk garbage lines at the start - and a couple at the end too. Let's go into COMAL and get rid of the first 4 junk lines.

(switch into COMAL now)

Run this program against the file:

```
0010 dim t$ of 99
0020 open file 2,"temp",read
0030 for junk=1 to 4 do input file
     2:t$ // Wrap Line
0040 open file 3,"fixed",write
0050 while not eof(2) do
0060   input file 2: t$
0070   print t$ // on screen
0080   print file 3: t$
0090 endwhile
0100 close
```

That got rid of the first 4 junk lines.

Now you can ENTER the lines into COMAL:

```
ENTER "FIXED"
```

You will get an error at the end (the word READY. is in the file at the end)--- Don't worry. Just hit the stop key (in COMAL 2.0) or SHIFT RETURN (in 0.14) to quit entering the program. Do a LIST - there it is. SAVE it QUICK before the power goes out.

Variation:

You can enter the TEMP file into EASYSCRIPT (or PaperClip via CNTRL J). Once in the word processor - just delete the garbage lines. Then save the file as a normal SEQ type file.

HINT !!!

Now you know - you can edit COMAL programs with a word processor if it can store the file back to disk as an SEQ type file. Make sure to keep the line numbers intact. Just use ENTER to get the program back into COMAL.

Hope this helps. If not, post a supplemental question!

Regards ... Len Lindsay

SUBJ: BASIC to COMAL
FROM: Captain C 06/23/86 S#: 27898

We plan to publish a COMAL 0.14 program (actually a two program set) that will convert BASIC programs into COMAL programs - plan on it for COMAL TODAY #13. It comes from Sol Katz.

Regards ... Len Lindsay

STRING FUNCTIONS

SUBJ: string func? (R6)
FROM: Icarus 04/24/86 S#: 7433

Can someone explain line 180 of Program 17 on page 81 of the Tutorial Binder. I know what it does. I added a line to print a$ so I could see it work in action. But I cannot figure out exactly how the code works, and I

More ►

am particularly puzzled by the placement of the colons.
Thanks.
Icarus (Jim V.)

SUBJ: strings (R)
FROM: Captain C 04/25/86 S#: 7815

User defined string functions are really handy. But since BASIC doesn't have them, they are unfamiliar to most.

Compare them to the built in string functions. You can talk about SPC$(5) without much trouble - it simply returns 5 spaces. So

```
A$=SPC$(5)
```

now A$ is equal to 5 spaces. Or

```
PRINT "HI",SPC$(5),"JIM"
```

That prints

```
HI     JIM
```

Now, we can make up our own string function:

```
FUNC jim$(num) CLOSED
  DIM temp$ OF num*3
  temp$="" // start with nothing
  FOR x=1 TO num DO temp$:+"JIM"
  RETURN temp$
ENDFUNC jim$
```

Now, try it:

```
PRINT jim$(3)
```

```
JIMJIMJIM
```

Now, about the colons ---

Colons are used to specify substrings

```
A$="BIRTHDAY"
```

A$(1:3) would be "BIR"
A$(2:5) would be "IRTH"

```
LONG=LEN(A$)
```

LONG is now 8

```
A$(LONG)  would be "Y"
```

If you don't specify both the start and end character - COMAL assumes you want only one character - the one at the position specified. In this case it was the same as

```
A$(8)
```

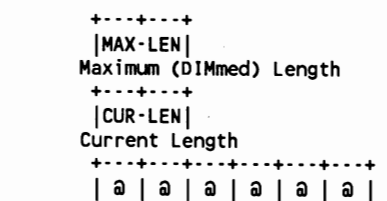
The colon separates the substring start postion from the end position.

SUBJ: More on colons: (R)
FROM: COMALite J 04/25/86 S#: 8472

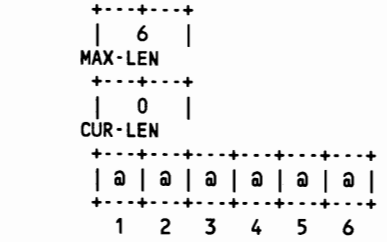
Another way to look at the colons is to think of just what a string actually is -- a one-dimensional array of characters. So, if I execute:

```
DIM mystring$ OF 6
```

I set up an area of memory like this:



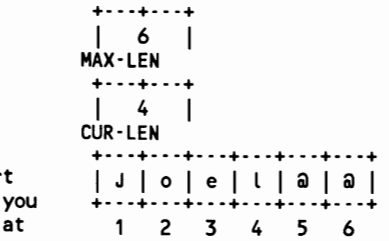
At first, it looks like this:



If we now execute:

```
mystring$:="Joel"
```

we get:



A string's current length can never exceed the maximum length specified in the OF clause of the DIM statement.

Unlike a numeric array, a string can have an apparent current dimension that is smaller than its actual maximum dimension. Also unlike a numeric array, you can reference the entire string or any contiguous part of it containing 0 or more characters with indexing. That sounds much harder than it is. Instead of a single number specifying which item of a numeric array to index, we use a RANGE to specify a SUBSTRING of a string. A RANGE may be in the format:

```
<range> ::= ( <lower bound>
            : <upper bound> )
```

where <lower bound> and <upper bound> are both numeric expressions which evaluate into a range of from 1 to MAXLEN. The range appears just after the string expression it references. So:

```
PRINT mystring$(2:3)
```

would PRINT characters 2 through 3 of mystring$ (currently "Joel"), thus "oe".

The expression "mystring$(2:3)" does this:

Suppose I now say:

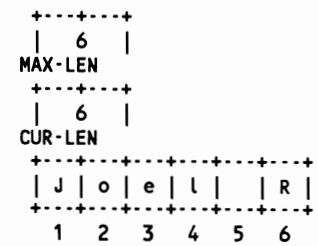
```
mystring$:+" Rea"
```

The ":+" assignment operator (COMAL 2.0 only) appends the specified string expression onto the end of the specified string variable. This is considerably faster and more memory- efficient than (as used in COMAL 0.14):

More ►

```
mystring$:=mystring$+" Rea"
```

but the end results are the same. The results would normally be a string with the value "Joel Rea", but since the string was DIMensioned with a MAX-LEN (maximum length) OF 6, the characters after the 6th character are truncated off. So the result is "Joel R", my screen name on PlayNet. Thus:



Now if I type:

```
PRINT mystring$(2:4)
```

the result would be "oel", namely characters 2 THROUGH 4 of the string "Joel R". If I just want character 3, I can specify the <lower bound> and the <upper bound> to both be 3, thus:

```
PRINT mystring$(3:3)
```

would just PRINT "e", the third character.

COMAL provides us a shortcut. If we want only one character (i. e. both the <lower bound> and the <upper bound> are equal), we can specify only the <lower bound> and leave off both the colon and the <upper bound>. So, the above can be abbreviated to:

```
PRINT mystring$(3)
```

Also, if you want to reference a substring starting with the first character of a string, you would normally specify a <lower bound> of 1. COMAL 2.0 gives you another shortcut: simply omit the <lower bound> of 1, and just specify the colon and the <upper bound> instead. So:

```
PRINT mystring$(:4)
```

is equivilant to:

```
PRINT mystring$(1:4)
```

and both PRINT "Joel". In addition, if you want your substring to end with the last character of the string, your <upper bound> would be equal to CUR-LEN, or "LEN(string'variable$)". In this case, we know that "mystring$" is 6 characters long at present, but in a program we don't always know that. So instead of having to specify a LEN function each time, you need specify only the <lower bound> and the colon. Thus:

```
PRINT mystring$(3:) // 2.0 only
```

is equivilant to:

```
PRINT mystring$(3:LEN(mystring$))
```

which, in this case, is equivilant to:

```
PRINT mystring$(3:6)
```

all of which PRINT "el R".

Indexing is much superior to Microsoft BASIC's form of string handling with LEFT$, MID$ and RIGHT$ functions. If you want them, though, here they are:

```
FUNC mid$(strng$, strt, lngth)
  RETURN strng$(strt:(lngth+strt)
  -1) // wrap line
ENDFUNC mid$

FUNC left$(strng$, lpos)
  RETURN mid$(strng$, 1, lpos)
ENDFUNC left$
        -- OR --
FUNC left$(strng$, lpos)
  RETURN strng$(:lpos)
ENDFUNC left$

FUNC mid2$(strng$, mpos)
  RETURN mid$(strng$,mpos,(LEN(
  strng$)-mpos)+1) // wrap line
ENDFUNC mid2$
        -- OR --
FUNC mid2$(strng$, mpos)
  RETURN strng$(mpos:)
ENDFUNC mid2$

FUNC right$(strng$, rpos)
  RETURN mid2$(strng$, LEN(strng$)
  -rpos) // wrap line
ENDFUNC right$
        -- OR --
FUNC right$(strng$, rpos)
  RETURN strng$(LEN(strng$)-rpos+1:)
ENDFUNC right$
```

There is one VERY nice feature that BASIC just can't do that COMAL can: in COMAL, you can ASSIGN to a SUBSTRING!! For example, assuming "mystring$" still equals "Joel R", then:

```
mystring$(2:4):="OEL"
PRINT mystring$
```

would print "JOEL R"! Characters 2-4 were changed without affecting the rest of the string! Substrings are padded with spaces to the right if needed:

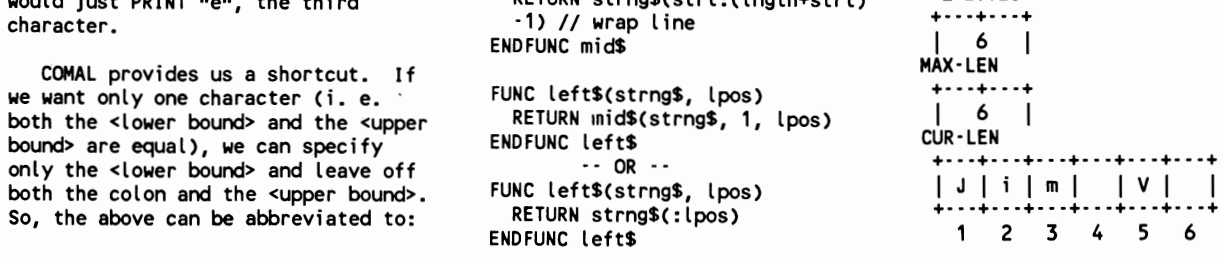
```
mystring$(2:4):="oe"
PRINT mystring$
```

PRINTs "Joe R", and:

```
mystring$(2:):="im V"
PRINT mystring$
```

PRINTs "Jim V "!

You will notice that in my stringstorage diagrams in Parts 1 and 2, I showed MAX-LEN and CUR-LEN as taking TWO bytes:



There was a reason: BASIC doesn't use a MAX-LEN, only a CUR-LEN, so BASIC needs "garbage collections" which can pause your program for long minutes to keep strings away from other data. COMAL strings know their place and stay put!

But there's more: BASIC's CUR-LEN only uses 1 byte. Since 1 byte can only hold a number in the range 0:255, a string in BASIC is limited to a maximum of 255 characters. 2 bytes can hold a number in the range 0:65535. Since free memory, even

More ►

under COMAL 2.0, is WELL under that, a COMAL string can be as big as you want to DIMension it to: up to all of free memory if you want!

Imagine: a word-processor in COMAL. The entire edit-buffer could be a single string variable!

On that note, I'll call it quits. Hope I was able to help!

SUBJ: string func?
FROM: Icarus 04/27/86 S#: 10369

Joel--
Thanks. That was quite a lesson. Now if there were only a way to capture the text in a buffer for storage on disk and hardcopy.... I will experiment some more with string functions. It's a good topic for a series of articles. (hint)
Have you got an amiga yet? Hope to see you soon. Thanks again.

BACK UP COMAL DISKS

SUBJ: Backing up Comal Disks (R2)
FROM: QueenB8354 04/21/86 S#: 4540

Recently purchased disks directly from Comal USA: Programmers Paradise Disk with Best of Comal on back side, and Auto Run Demo with Tutorial on backside. Attempted to back these up using Copy program on Paradise disk; only Tutorial copy will run same as original. Others appear to have incomplete directories. Should I return these originals to Comal they appear to work fine but I dislike using the originals and really want backups. Even attempted back up with another copy program with same/similar results. PLEASE HELP! Getting desperate. Thanks.

SUBJ: Backing up COMAL disks (R)
FROM: COMALite D 04/21/86 S#: 4842

The problem you are having is one that was recently discovered. Actually the disk AND the backup program are fine. All the files are correct, and can be run properly. The problem is that the commercial program "David's Directory Designer" was used to insert comments into the directory (making it more readable, and pointing out programs or files that cannot be LOADed or RUN). Unfortunately this program DOES NOT update the section of the disk that tells what blocks of the disk are in current use ("allocated").

Adding the comments as file entries resulted in more blocks being used, and the new blocks were not "allocated" to show that they were needed. The program "backupdisk.basic", along with many other copy programs, will only copy the blocks that show that they are needed (are allocated).

The "fix" for this problem is to add a line to the program "backupdisk.basic" (enter it as one long line):

```
2625 IFT%=18THENFORJ=1TO8:BM%(18,S-J)
     =0:NEXT:REM WRAP LINE
```

Just enter the following commands from BASIC:

```
NEW
LOAD "backupdisk.basic",8
<add line above>
RUN
```

The added line will automatically force the program to copy the entire directory, not just the blocks that show that they are in use. I would suggest that you SAVE the program with the new line on another disk and delete the "backupdisk.basic" program on the copy and replace it with the "fixed" version.

This problem may exist with EVERY disk put out by COMAL Users Group for the last six months. It is only a problem for those users who lack fast backup programs which do the entire disk. Future disks should not have this problem.

SUBJ: BACKUP FIX
FROM: Captain C 06/23/86 S#: 27903

The fix for the backup program will be published in COMAL TODAY #13.

Regards ... Len Lindsay

DIRECTORY READING

SUBJ: Disk Directory (R2)
FROM: MikeR1 04/13/86 S#: 61089

Well, I am in a PICKLE!! As you'll notice in my previous message I was wondering if I could use that copyrighted program in mine. Well, just through curiousness, I tried to ENTER it to my program called ViewDOS (which is nearly done). Well, it merged without hitch, BUT when I tried to run it OH NO!!! After it dimensioned, there was no ROOM left. I have the 11K version of 0.14 running so I can't expand it any further. Does ANYONE have a disk directory printer (to screen or printer or both) that doesn't take up as much memory (and if possible is FASTER than the other)? Thanks! ViewDOS is ALMOST done! I just have to add the printer functions to it and create/modify some support software!

Thanks for any information you can tell me about a DIRECTORY printer!

SUBJ: DIR in 0.14 (R)
FROM: COMALite D 04/14/86 S#: 61437

I wrote a short (15 lines) procedure that emulates the DIR command of COMAL 2.0 (does pattern matching). It was printed in COMAL TODAY #10 (Page 28).

Here is a listing of the procedure:

```
proc dir(name$,device) closed
 trap esc-
 for x:=1 to len(name$) do
  poke 834+x,ord(name$(x))
 endfor x
 poke 26997,x
 poke 27013,device
 poke 27110,96
 sys 26996
 poke 27110,76
 poke 26997,1
 while esc do null
 trap esc+
endproc dir
```

You will notice that the procedure actually changes some bytes in COMAL 0.14 and calls the CAT command. This makes the procedure short, but DOES NOT allow printing the directory to the printer. For that you will have to have a separate program that reads the directory into memory, and not the screen.

This procedure will also allow you to press the CTRL key to slow the listing or press the STOP key to abort it (without stopping the program).

More ►

If anyone wants to find out what changes could be made in the COMAL CAT command code to output to the printer, feel free to investigate the code around 26990 (decimal). If you find the necessary changes, please post them here or in the COMAL 0.14 area.

SUBJ: directory printer
FROM: Icarus 04/19/86 S#: 3138

This is a wild idea. Could you use the ML program that makes a proc DIR (in CT 10 or 11 I think) and modify it for output to printer? Or even do a select lp: first to see what happens? I haven't tried it. But it is not a long program, so if it can be modified it might work for you.
Good luck.
Icarus

SUBJ: Directory Printer (R1)
FROM: MikeR1 04/12/86 S#: 60294

I was wondering if the (C) program written by the COMAL USERS GROUP that will print the directory to the screen/printer can be used in my own COMAL programs. If not, are there any directory printers out there that I can get a hold of?

SUBJ: Directory Printer
FROM: Captain C 04/15/86 S#: 61827

All of our directory printer programs and routines are available for you to use in your programs and on your disks.

I'm not aware of any copyrighted directory programs from us.

The COMAL 0.14 system itself is copyrighted - and we give you permission to make copies of it for your own backup - as well as to give away copies to your friends and schools - but not to sell.

Some of our special disks are copyrighted and we request that these special disks be only copied for your own backup purposes. Others who wish to have these disks should get them directly from COMAL Users Group USA.

Thanks ... Len Lindsay

PRINT USING

SUBJ: edit patterns (R3)
FROM: Steve mc 04/03/86 S#: 55762

Is there a way to print using a pattern that will NOT suppress leading zeros?

SUBJ: print using (R)
FROM: Captain C 04/04/86 S#: 55961

PRINT USING always supresses the leading zeros.

It is possible to write a routine that will include the zeros. It could be a closed function or procedure that could then be merged onto any program.

If anyone has such a routine, please post another reply to this.

Thanks ... Len Lindsay

SUBJ: PRINT USING (R)
FROM: Captain C 04/04/86 S#: 55962

There is a LONG message from a few weeks ago explaining the use of PRINT USING. Check it out for a full explanation.

SUBJ: Try this:
FROM: COMALite J 04/04/86 S#: 56396

For COMAL 2.01 only:

```
FUNC pad'it$(number,length,pad$)
  CLOSED // wrap line
  DIM p$ OF length, n$ of length
  FOR i#:=1 TO length DO lz$:+pad$(1)
  n$=STR$(number)
  p$(:length-LEN(n$)):=n$
  RETURN p$
ENDFUNC pad'it$

Examples:

 SCAN

 PRINT pad'it$(45,5,"0")
 00045

 balance'due:=123.45

 PRINT "$"+pad'it$(balance'due,9,"*")
 $***123.45
```

NOTE:

It will be somewhat harder for COMAL 0.14, due to the lack of user-written string FUNCtions (can get around by using a PROCedure and returning the result in a REFerenced string parameter.) and STR$ built-in FUNCtion (now available in COMAL 0.14+).

COMALite J (Joel Ellis Rea)

1520 PLOTTER

SUBJ: output to devices (R1)
FROM: BLC 03/17/86 S#: 47906

When I first saw COMAL I was very impressed and decided to forget BASIC, then ordered the handbook. My prime objective was to write some plotting utilities for the Commodore plotter.
Unfortunately though, for some reason the system sends a space out on the bus before each print. The plotter, expecting a specific set of commands, does not respond.

My question then is: How do you inhibit the preceeding space generated in print statements?

If anyone knows, please respond.

THANKS.

SUBJ: 1520 Plotter - no problem
FROM: COMALite D 03/18/86 S#: 47967

I don't completely understand the problem you are having. COMAL communicates with the plotter fine. We printed a very nice 2 page program listing in COMAL TODAY #7 (page 62) that was a good set of procedures for easy control of the plotter.

I think your problem may result from the use of semi-colons in your print statements - The manual says to use them from BASIC - but the equivalent in COMAL is the COMMA. Perhaps just change your semicolons into commas and all will be well.

The plotter is device number 6. You can open a file to the device like this:

```
sa=0
open file 2,"",unit 6,sa,write
print file 2: "hello"
close file 2
```

The sa is the secondary address

More ►

needed. Each secondary address has specific meaning to the plotter.

Or you can set up the plotter to accept output just as if it were a printer. The default printer selection goes to unit 4 - we need to open file 255 (the printer file) immediately before the select command to override this default like this:

```
open file 255,"",unit 6,0,write
select output "lp:"
print "hi there"
list
select "ds:"
```

The top two lines select the plotter for your output ("lp:" means line printer - but we overrode that in the first line). Any program in the computer would be listed to the plotter using the above commands.

Any specific questions please post them here. If you have COMAL TODAY #7 check out the article and example listing.

Programs that use the plotter are included on several of our disks including:

TODAY DISK #7
TODAY DISK #9
USER GROUP DISK #10
UTILITIES SET #2

I will upload an example plotter program for you to check out (allow a couple days for it to be "verified" by QLINK).

Other COMALites with extra help are encourage to post a response to this message - or another reponse to the original.

Regards ... Len Lindsay &
David Stidolph

PRINT USING

SUBJ: PRINT USING
FROM: Captain C 03/04/86 S#: 41564

Several people have requested info on PRINT USING, so here it is (briefly!):

PRINT USING is a built in command in COMAL. The simple form is:

```
PRINT USING <format string>:values
PRINT USING"Total Due: $###.##":total
```

If issued three times with these values (2,243.51,15.5) it would print:

```
Total Due: $  2.00
Total Due: $243.51
Total Due: $ 15.50
```

It's use with money is obvious, since it easily lines up the decimal points. It also rounds the numbers for you automatically if the "format" is exceeded. Value 3.1234 in above prints:

```
Total Due: $  3.12
```

Nice and neat. However, it can be expanded to several numbers:

```
PRINT USING"##> $###.##":number,total
```

Could print:

```
 8> $899.95
 9> $ 55.01
10> $  0.04
```

The "format" could be a variable:

```
dim format$ of 30
format$="Pay me $###.## right now."
amount'due=589
print "OK. This is enough."
print using format$:amount'due
```

This would print like this:

```
OK. This is enough.
Pay me $589.00 right now.
```

You also can mix variables and constants for the format string, and the values can be constants as well as variables too. ■

COMAL Structures FOR Loops

By Richard Bain

The **FOR** loop is the one structure that people coming from BASIC may already know. Whether you have used a **FOR** loop before or not, the following tips may be helpful. In COMAL, a loop groups statements which need to be repeated together. The **FOR** loop is a unique COMAL structure in that the number of times the loop executes is specified explicitly in the program code. A counter automatically keeps track of how many times the loop has repeated.

The following example reads in and prints out the names of 3 people along with their phone numbers. It is given first without a **FOR** loop, and then is updated to use one. Type in the common code first; then type in either method 1 or 2, but not both together.

common code:

```
0010 data "John Doe","555-1111"
0020 data "Jane Doe","555-2222"
0030 data "Mr Smith","555-3333"
0040 people:=3
0050 dim name$(people) of 10, phone$(people) of 8
```

method 1:

```
0060 read name$(1),phone$(1)
0070 print name$(1);phone$(1)
0080 read names(2),phone$(2)
0090 print name$(2);phone$(2)
0100 read names(3),phone$(3)
0110 print name$(3);phone$(3)
```

method 2:

```
0060 for x:=1 to people do
0070   read name$(x),phone$(x)
0080   print name$(x);phone$(x)
0090 endfor x
del 100 - 110
```

In this example, method 2 is shorter, but that is not its main advantage. In the future, you may want to add more names to your list. Using either method, you must add the **DATA** statements and change the number that **people** is set to. In method 1, you must also add more **READ** and **PRINT** statements; in method 2, nothing more needs to be added. (The idea of being able to maintain or update programs is not important for beginners; it is hard enough to get a program to work. However, the intermediate programmer should begin to think ahead. That is why a variable, **people**, is more useful than a 3.)

Sometimes you may only want to repeat a single statement. There is a special kind of **FOR** loop to let you do just that. In the above example, you might want to **PRINT** the **DATA** again. This could even be done from the immediate mode. Here is how to do it with a one line **FOR** loop (Note that **ENDFOR** is not used):

```
for x:=1 to people do print name$(x);phone$(x)
```

FOR loops can be nested, one inside another. This example prints a multiplication table:

```
for row:=1 to 10 do
  for col:=1 to 10 do print using "###":row*col;
  print
endfor row
```

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

More ►

For Advanced Programmers Only

The **FOR** loop is fairly easy to master, but there are a few things to look out for. One is assigning a value to the counter. Some might be tempted to force the loop to end by assigning the ending value to the counter. This works, but is bad programming when you have so many types of loops to choose from. A more dangerous side effect of assigning a value to the counter is in creating infinite loops. This can happen if you assign a value to the **FOR** loop counter which is between the starting and ending values.

A second problem to look out for in **FOR** loops is roundoff error. The **FOR** loop is extremely sensitive to roundoff error if the counter is a real variable and the stepsize is not an integer. Look at the following example:

```
for x:=0 to 13 step 13/3 do print x
```

This will print out the numbers 0, 4.33, and 8.67, but what about 13? The counter will be set to 13 plus or minus any roundoff error. If the roundoff error makes x greater than 13, say 13.000000003, then the loop will stop before 13 is printed. If there is no roundoff error or the error is negative, then 13 will be printed. This uncertainty is unacceptable in a structure designed to execute a fixed number of times. If the above numbers are needed, there is a safer way to write the loop:

```
for x:=0 to 3 do print 13*x/3
```

Much to my surprise, I found that COMAL 0.14 prints the 13 in the original example and COMAL 2.0 does not. Apparently, they use different floating point routines? ■

Best Sellers

COMAL Today - The Index is now in stock! It is already up to #2 on the list. The *Index* is the same size as a *COMAL Today* and will fit in nicely with all your issues. It has 4,848 entries in 56 pages. Best of all - the first printing has a very special cover - printed on **Silver Currency** heavy paper. Future printings will be on the normal Glossy White stock we use for *COMAL Today*. Get your *Index* right away and get the **Silver Collectors Item.**

The two all time favorite COMAL books according to those returning the **vote** postcard in *COMAL Today #14* are *COMAL Handbook* and *Cartridge Tutorial Binder.* Make sure you return the **vote** postcard in this issue! You can even use it as a "coupon"!

September 1986

#1 - COMAL Workbook
by Gordon Shigley
#2 - Cartridge Tutorial Binder
by Frank Bason & Leo Hojsholt
#3 - Foundations in Computer Studies
by John Kelly
#4 - COMAL From A to Z
by Borge Christensen
- Cartridge Graphics and Sound
by Captain COMAL's Friends
#5 - COMAL Today - The INDEX
by Kevin Quiggle

October 1986

#1 - COMAL From A to Z
by Borge Christensen
#2 - COMAL Today - The INDEX
by Kevin Quiggle
#3 - COMAL 2.0 Packages
by Jesse Knight
#4 - Cartridge Graphics and Sound
by Captain COMAL's Friends
#5 - COMAL Handbook
by Len Lindsay ■

Kaprekar

by Jack Baldrige

One unimportant but interesting number, which I first heard of as Kaprekar's constant, could best be described as a footnote to mathematical number theory.

For quite a few years, when learning a new computer language, I would write a program to illustrate this footnote. I recently saw it mentioned in a column on mathematical recreations in *Byte* magazine (August, 1986). I suddenly remembered that I had written a demo while I was very new to COMAL. Perhaps you will enjoy it too.

To start with, take an integer number from 1 to 9998: the only restriction is that it can't have all four digits the same. From its digits, make the largest and smallest number. *3749* creates *3479* and *9743*. Then subtract the smaller one from the larger. When this procedure is followed not more than seven times, you will find a constant difference, Kaprekar's constant. This program is on *Today Disk #15*.

```
intro
repeat
  count:=0
  print " Input a 1-4 digit integer,"
  input " not all digits the same:": num1#
  repeat
    count:+1
    break'it'down
    sort(n#)
    build'it'up
    num1#:=num3#-num2#
    if num1#=0 then
      print " Input Data Error"
      end
    endif
    print using "##   ": count,
    print using "####-": num3#,
    print using "#### = ": num2#,
    print using "####": num1#
  until num1#=6174
  input " Hit RETURN to continue, (Q to Quit)": opt$
until opt$ in "Qq"
//
proc break'it'down
  num2#:=num1# mod 100; num1#:=num1# div 100
  n#(4):=num1# mod 10; n#(3):=num1# div 10
  n#(2):=num2# mod 10; n#(1):=num2# div 10
endproc break'it'down
proc sort(ref n#()) closed
  for top:=3 to 1 step -1 do
    for k:=1 to top do
      if n#(top+1)>n#(k) then swap(n#(top+1),n#(k))
    endfor k
  endfor top
endproc sort
//
proc swap(ref j#,ref k#) closed
  m#:=j#; j#:=k#; k#:=m#
endproc swap
//
proc build'it'up
  num2#:=0; num3#:=0
  for k:=0 to 3 do
    num2#:+n#(k+1)*10
    num3#:+n#(4-k)*10
  endfor k
endproc build'it'up
//
proc intro
  dim n#(4), quit$ of 4, opt$ of 1
  opt$:=chr$(0); quit$:="eExX"
  print chr$(14),"      From a 4-digit integer, not a"
  print "   multiple of 1111, form the largest"
  print "   and smallest integers which can be"
  print "   made from its four digits, and find"
  print "   and find the difference."
  print "      Repeat this process. The answer"
  print "   soon becomes constant and equal to"
  print "   6174, Kaprekar's constant."
  print "         Press any key to Continue "
  while key$=chr$(0) do null
endproc intro
```

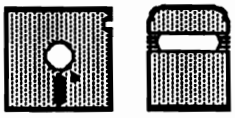
■

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

ROTATE

This game by Dick Klingens starts with 16 little squares, each containing a letter of the alphabet. These little squares make one big square. The object of the game is to rotate medium size squares, 4 little squares each, until the letters become arranged in alphabetical order. To rotate a medium size square, type in the letter from the upper left hand corner of that a square. Due to the nature of the game and the method of initializing the square, it might not be possible to solve the puzzle solely by rotating medium size squares. Therefore, you are allowed to exchange two squares which are side by **once** during the game. When you have solved the puzzle, the computer will tell you how many turns you have taken. *Rotate* is on *Today Disk #15*. ■

PRINT FILE Numbers

by Captain COMAL 

Riddle: When is a 4 not a 4 ?
Answer: When it is written by BASIC.

Paul Stanko noted that BASIC and COMAL differ in printing numbers. Here is what we discovered.

When BASIC prints a number, it actually prints a space, followed by the number, followed by another space. Take the following small BASIC program:

```
10 open2,8,2,"numbers.basic,s,w"
20 for x=1 to 3
30 print#2,1","2","3","4","5","6
40 next
50 close2
```

It creates a file that looks like this:

```
 1 , 2 , 3 , 4 , 5 , 6
 1 , 2 , 3 , 4 , 5 , 6
 1 , 2 , 3 , 4 , 5 , 6
```

All those extra spaces are put in by BASIC whether you like it or not. It's been a long time since I've used BASIC, but I do remember **not** liking those spaces.

However, COMAL prints a number as just the number. We all are glad about that, but what can we do with numeric data files written by BASIC so that COMAL can read them? As you might have guessed, a very short COMAL 2.0 program will convert the numeric data file into one readable not only by COMAL, but by BASIC as well!

After running the following conversion program, our data will look like this:

```
1,2,3,4,5,6
1,2,3,4,5,6
1,2,3,4,5,6
```

```
0010 open file 2,"0:numbers.basic",read
0020 open file 3,"0:numbers.dat",write
0030 while not eof(2) do
0040   char$:=get$(2,1)
0050   if char$<>" " then
0060     print file 3: char$,
0070     print char$,
0080   endif
0090 endwhile
0100 close file 2
0110 close file 3
```

Since the conversion program uses **GET$**, a COMAL 0.14 version of it must include our **disk'get** routines (listed in the *COMAL Handbook*). The COMAL 0.14 version is on *Today Disk #15.*

Once our example data file has been converted, the following BASIC program can still read it:

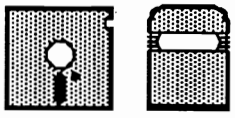
```
10 open2,8,2,"numbers.dat,s,r"
20 for x=1 to 3
30 input#2,a,b,c,d,e,f
40 print a,b,c,d,e,f
50 next
60 close2
```

This COMAL program can also read it:

```
0010 open file 2,"numbers.dat",read
0020 for x:=1 to 3 do
0030   input file 2:a,b,c,d,e,f
0040   print a;b;c;d;e;f //; gives a space
0050 endfor x
0060 close file 2
```

Of course, if you are just using COMAL, please use **WRITE FILE** and **READ FILE** instead of **PRINT FILE** and **INPUT FILE.** They are more efficient, faster, and less prone to cause you problems! Really! UniCOMAL themselves recommend it. ■

Doctor Who - The Data Base



by Len Lindsay

This issue I wanted to have several programs demonstrate file handling, especially with random files. **Doctor Who** presented itself as an excellent topic. This science fiction show has been on the air for 23 years now, with a total of over 150 movie length shows. In each show, the Doctor has varying companions, and meets different adversaries (and even interesting monsters). So, I created a data base system that can easily keep track of a series of shows. Very little in this data base is specific to **Doctor Who**. In fact, next issue we hope to show how to adapt it to be a **Star Trek** Data Base System. I used 8 fields of 27 characters each to store the data for a show, plus two extra fields that you can decide how to use. The titles for each of the 8 main fields can vary depending on what information you wish to keep track of. For my **Doctor Who** shows I chose the following:

Show Name:
Doctor:
Companion1:
Companion2:
Companion3:
Adversary1:
Adversary2:
Location:

I've also provided two extra "user" fields. The first is an "ID" of 11 characters. You could put the video tape number here, the name of your friend who has this show on tape, or a short comment on the show. The other field is only one character. It can be used to rate each show (A,B,C, etc). I use it to mark the shows that I have seen.

The program listing that follows this article is the complete data base program.

However, the data base itself is empty! Since I am a true **Doctor Who** fan, I have painstakenly entered all the data for the first **144** shows! This large data base of information is on *Today Disk #15* along with the data base program itself (an enhanced version for COMAL 2.0 is on the 2.0 side of the disk). Once you have a data base filled with information, it is easy and fun to use.

For example, let's say your favorite adversaries were the **CYBERMEN**. Using the data base system, you can choose the **S**earch option, and it will list every show that included cybermen as adversaries! The nice thing about this is that the system uses COMAL's **IN** operator to find the matches. Thus, you can ask the system to look for **CYB** and it will find both CYBERMEN as well as their "pets", the CYBERMATS. Remember, the search is very picky on upper or lower case. Thus, **Cyb** will not match **CYB**. Therefore, you may wish to do as I did on my data -- never use shifted letters! To help you remember, the first thing the program does is flip into uppercase / graphics characters -- and lock it in by disabling the SHIFT / Commodore key case toggle.

Since this system uses a random file for its data, it is easy to randomly display any show. Just type in the show number, and instantly (almost) its information is displayed. To see the next show just type + (plus sign). To see the previous show type - (minus sign).

Another nice feature of this system is **BROWSE**. It allows you to scan over all the entries, one after another, starting after the show currently displayed. It even has a variable pause between shows. Five seconds is the default, but for a fast scan, use 0 for the length of pause.

More ►

NOTES ABOUT THE PROGRAM

The variable **mark$** is used to determine whether or not the information display is normal or reverse field. The seventh line of the program is:

```
display(mark$<> " ")
```

Display is the name of the procedure that prints the information about the show to the screen. It has one numeric parameter. If the parameter is **FALSE** (a value of 0) then the information is displayed normally. If it is **TRUE** (a value not equal to 0) then the information is displayed in reverse field. **Mark$<> " "** is a comparison that will always be either **TRUE** or **FALSE**, thus providing the proper parameter.

Notice that to have the reverse display, each value printed must be the correct length because the reverse field will stop at the last character. The length used in this program is 27. Thus, if the show name is less than 27 characters, I had to do something to print reverse field spaces at the end of the name. I let COMAL take care of this for me by using substring notation when inputing this data (from both keyboard input and file input). For example:

```
dim name$ of 27
input "name:": name$
print chr$(18)+name$
print "length is";len(name$)
input "name:": name$(1:27)
print chr$(18)+name$
print "length is";len(name$)
```

Run that program and type **TEST** as the name both times it asks. The first time it has a length of 4. The last time it has a length of 27 (the value was padded with spaces at the end). Printing it each time in reverse field shows how the (1:27) takes care of formatting problems for me!

Calculating the correct record length is also important when using random files. Use the **maximum** possible length as the length for the file. In this case I had 8 fields of 27 characters, 1 field of 11 characters, and 1 field of 1 character. Since each field is a string, I also must add a 2 byte counter for each field. Thus 8*29 plus 13 plus 3 gives 248. The smallest record size I can use and be sure to hold all possible data is 248, which I set in the start'up procedure.

To calculate record length when using **WRITE FILE** remember these rules:

* A real number always needs 5 bytes
* An integer needs 2 bytes in COMAL 2.0
  An integer needs 5 bytes in COMAL 0.14
* A string needs its maximum length
  PLUS 2 bytes for a length counter

Knowing how many records have been written is also important when using random files. If you attempt to read a record past the last existing one, an error occurs! A common way to keep track of the last record number is to WRITE it into the first record of the file:

```
write file 2,1: last'record
```

Last'record is a variable that holds the number of the last record written to the file. Now, each time you use that file, you can simply read the last record number from the file like this:

```
read file 2,1: last'record
```

The only side effect of this is that you cannot use the first record for regular data. Often, this is not a problem. For

More ►

example, in our order processing system programs (running under IBM PC COMAL) *COMAL Today* subscribers are part of a random file, stored by their subscriber number. Since I use the first record to hold the last subscriber's number, I started my subscriber count with 2. Thus there is no subscriber number 1.

But in keeping track of **Doctor Who** shows, there **is** a show number 1. Since it can't be written into record number 1, we simply add 1 to the show number to give the record number (read this sentence twice - it makes sense). Thus show number 1 is stored in record number 2. Both **read'record** and **write'record** procedures do the record number conversion.

Another thing I try to do is keep my files closed as much as possible. On a Commodore disk, all files must be properly closed or they can't be opened later. So, I take precautions (in case of a power outage etc.) and keep the file closed when it is not directly being used. For example, while **searching** the file for matches, I use the procedure **read'record** directly, and don't close the file after each record is read. I only close the file after I find a match. However, when just getting information about one show, I have the procedure **read'it** which first opens the file, then reads the record, then closes the file. Thus the file is closed while I look at that record (and possibly edit it). Note that I always close the file after I write to it. Thus I only have one **write'record** procedure.

Remember, the data is one of the most important parts of this data base system! And we can't list that here - **it is 144 blocks on the disk**! The same data file is used by both the 0.14 and 2.0 versions of this data base program.

COMAL 2.0 Users Notes

This program works in both COMAL 2.0 and 0.14, but only the 0.14 version is listed. Both versions are on *Today Disk #15*, complete with a data base full of information that took hours to enter. The 2.0 version on the disk includes several enhancements over the 0.14 version listed here. However, the program as listed should work in 2.0 if you keep the following notes in mind:

* PAGE is part of COMAL 2.0. Do not type in the PAGE procedure.
* CREATE is part of COMAL 2.0. Do not type in the CREATE procedure -- and remove the () parentheses from the CREATE statement (in procedure start'up).
* CURSOR is part of COMAL 2.0. Do not type in the CURSOR procedure -- and remove the () parentheses from the CURSOR statements.
* PRINT AT is part of COMAL 2.0. Do not type in the PRINT'AT procedure. Also remove the ' apostrophe and the () parentheses from each PRINT AT statement, and change the last "," comma into a ":" colon.
* Change all PENCOLOR to be TEXTCOLOR.
* Change all BACKGROUND to be TEXTBACKGROUND.
* Change all BORDER to be TEXTBORDER.
* At the beginning of the program add:
  USE system
  USE graphics
* COMAL 2.0 has a VAL function that can be used to improve the VALUE function in the program. You could change it to:

```
FUNC value(x$)
  TRAP
    RETURN VAL(x$)
  HANDLER
    RETURN 0
  ENDTRAP
ENDFUNC value
```

More ►

* FILE'EXISTS should be changed to the 2.0 method:

```
FUNC file'exists(filename$) CLOSED
  TRAP
    OPEN FILE 78,filename$,READ
    CLOSE FILE 78
    RETURN TRUE
  HANDLER
    CLOSE FILE 78
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
```

Version 0.14 - Dr. Who Data Base

```
// doctor who data base by len lindsay
print chr$(147)+chr$(142)+chr$(8)
start'up
clear'keys
repeat
 format'screen
 display(mark$<>" ")
 choices
until done
halt
//
proc start'up
 boxes:=0
 title'color:=1
 data'color:=7
 desc'color:=15
 dull'color:=9
 background (4)
 border (0)
 pencolor (1)
 page
 print "setting up ... please wait ..."
 dims
 filename$:="ran.doctorwho"
 record'length:=248 // 8*29 + 13+3
 if not file'exists(filename$) then
  print "file not found. hit stop to stop or"
  input "hit <return> to create it:": reply$
  create(filename$,144,record'length)
  current'show:=1; last'show:=1
  write'last
  add
 else
  read'last
  current'show:=1
  read'it(current'show)
 endif
endproc start'up
//
proc format'screen
 page
 pencolor (boxes)
 print "+-----------------------------------+"
 print "!";
 pencolor (title'color)
 print "    doctor who data base system    ";
 pencolor (boxes)
 print "!"
 print "+---------+----------+-------------+-+"
 print "!         !    of    !id:           ! !"
 print "+---------+----------+-------------+-+"
 pencolor (desc'color)
 print'at(4,2,"show num")
 print'at(6,2,"show name:")
 print'at(7,2,"doctor   :")
 print'at(8,2,"companion:")
 print'at(9,2,"        2:")
 print'at(10,2,"        3:")
 print'at(11,2,"adversary:")
 print'at(12,2,"        2:")
 print'at(13,2,"location :")
endproc format'screen
//
proc choices
 cursor(14,1)
 pencolor (boxes)
 print "+--+-----------------------+"
 for lines:=15 to 22 do
  print "!  !                       !"
 endfor lines
 print "+--+-----------------------+"
 pencolor (title'color)
 print'at(15,2,"a")
 print'at(16,2,"e")
 print'at(17,2,"b")
 print'at(18,2,"n")
 print'at(19,2,"p")
 print'at(20,2,"##")
 print'at(21,2,"s")
 print'at(22,2,"q")
 pencolor (desc'color)
 print'at(15,5,"add shows")
 print'at(16,5,"edit this show")
 print'at(17,5,"browse (auto viewer)")
 print'at(18,5,"next show (+)")
 print'at(19,5,"previous show (-)")
 print'at(20,5,"display this show number")
 print'at(21,5,"search")
 print'at(22,5,"quit")
 cursor(24,1)
 pencolor (title'color)
 input " your choice: ": reply$
 clear'choices
 done:=false
 case reply$ of
 when "a","A"
  add
 when "e","E"
  edit'it
 when "b","B"
  browse
 when "n","N","+"
  next'show
 when "p","P","-"
  previous'show
```

More ►

```
  when "s","S"
   search
  when "q","Q"
   done:=true
  otherwise
   temp:=value(reply$)
   if temp>0 and temp<=last'show then
    current'show:=temp
    read'it(current'show)
   endif
  endcase
  pause(1)
  clear'keys
endproc choices
//
proc display(highlight)
 rvs$:="" //null
 if highlight then rvs$:=chr$(18)
 pencolor (desc'color)
 print'at(4,2,"show num ")
 pencolor (data'color)
 cursor(4,12)
 print using "###": current'show
 cursor(4,19)
 print using "###": last'show
 print'at(4,26,id$)
 print'at(4,38,mark$)
 print'at(6,12,rvs$+show'name$)
 print'at(7,12,rvs$+doctor$)
 print'at(8,12,rvs$+companion1$)
 print'at(9,12,rvs$+companion2$)
 print'at(10,12,rvs$+companion3$)
 print'at(11,12,rvs$+adversary1$)
 print'at(12,12,rvs$+adversary2$)
 print'at(13,12,rvs$+location$)
endproc display
//
proc input'data
 pencolor (data'color)
 cursor(6,1)
 input " show name:": show'name$(1:27)
 input " doctor   :": doctor$(1:27)
 input " companion:": companion1$(1:27)
 input "         2:": companion2$(1:27)
 input "         3:": companion3$(1:27)
 input " adversary:": adversary1$(1:27)
 input "         2:": adversary2$(1:27)
 input " location :": location$(1:27)
 input " id       :": id$(1:11)
 input " mark     :": mark$(1:1)
endproc input'data
//
proc read'record(show'number)
 if show'number<1 or show'number>last'show then return
 record'number:=show'number+1//convert show# to rec#
 read file 2,record'number: show'name$(1:27)
 read file 2: doctor$(1:27)
 read file 2: companion1$(1:27)
 read file 2: companion2$(1:27)
 read file 2: companion3$(1:27)
 read file 2: adversary1$(1:27)
 read file 2: adversary2$(1:27)
 read file 2: location$(1:27)
 read file 2: id$(1:11)
 read file 2: mark$(1:1)
endproc read'record
//
proc write'record(show'number)
 if show'number<1 then return //invalid
 record'number:=show'number+1//convert show# to rec#
 clear'line(24)
 print'at(24,1,"writing record...")
 open'it
 t1$:=show'name$
 t2$:=doctor$
 t3$:=companion1$
 t4$:=companion2$
 t5$:=companion3$
 t6$:=adversary1$
 t7$:=adversary2$
 t8$:=location$
 write file 2,record'number: t1$,t2$,t3$,t4$,t5$,t6$,
 t7$,t8$,id$,mark$ //wrap line
 close'it
 clear'line(24)
endproc write'record
//
proc add
 current'show:=last'show
 done'adding:=false
 repeat
  format'screen
  print'at(4,2,"adding no")
  cursor(4,12)
  print using "###": current'show+1
  display'bottom(false) //empty
  repeat
   input'data
   add'status
  until data'ok or done'adding
  if data'ok then
   current'show:+1
   last'show:+1 //record accepted
   write'record(last'show)
   write'last
  else  // abort
   read'it(current'show) // refresh values
  endif
 until done'adding
endproc add
//
proc edit'it
 repeat
  if mark$<>" " then display(false) //clear reverse
  display'bottom(true) //with data
  input'data
  edit'status
 until data'ok or done'editing
 if data'ok then
  write'record(current'show)
 else  // aborted
  read'it(current'show) //refresh values
 endif
endproc edit'it
//
proc write'last
```

More ►

```
 open file 2,filename$,random record'length
 write file 2,1: last'show
 close file 2
endproc write'last
//
proc read'last
 open file 2,filename$,random record'length
 read file 2,1: last'show
 close file 2
endproc read'last
//
proc next'show
 if current'show<last'show then current'show:+1
 read'it(current'show)
endproc next'show
//
proc previous'show
 if current'show>1 then current'show:-1
 read'it(current'show)
endproc previous'show
//
proc dims
 dim show'name$ of 27, find'show'name$ of 27
 dim doctor$ of 27, find'doctor$ of 27
 dim companion1$ of 27, find'companion$ of 27
 dim companion2$ of 27
 dim companion3$ of 27
 dim adversary1$ of 27, find'adversary$ of 27
 dim adversary2$ of 27
 dim location$ of 27, find'location$ of 27
 dim text$ of 27
 dim filename$ of 18
 dim reply$ of 40, continue$ of 1
 dim t1$ of 27, t2$ of 27, t3$ of 27, t4$ of 27
 dim t5$ of 27, t6$ of 27, t7$ of 27, t8$ of 27
 dim id$ of 11, mark$ of 1, rvs$ of 1
 dim find'id$ of 11, find'mark$ of 1
endproc dims
//
proc open'it
 open file 2,filename$,random record'length
endproc open'it
//
proc close'it
 close file 2
endproc close'it
//
proc read'it(rec'num)
 open'it
 read'record(rec'num)
 close'it
endproc read'it
//
proc clear'choices
 for lines:=14 to 24 do
  clear'line(lines)
 endfor lines
endproc clear'choices
//
proc find'input
 format'screen
 display'bottom(false)
 pencolor (data'color)
 print'at(4,12,"all")
 cursor(4,19)
 print using "###": last'show
 pencolor (dull'color)
 print'at(4,27,"searching:")
 pencolor (title'color)
 print'at(24,1,chr$(18)+" enter text to search for ")
 pencolor (title'color)
 cursor(6,1)
 input " show name:": find'show'name$
 input " doctor   :": find'doctor$
 input " companion:": find'companion$
 print "         2:-------"
 print "         3:-------"
 input " adversary:": find'adversary$
 print "         2:-------"
 input " location :": find'location$
 input " id       :": find'id$
 input " mark     :": find'mark$
 clear'line(24)
endproc find'input
//
proc search
 find'input
 trap esc-
 open'it
 searching:=0
 continue$:="" //keep searching
 pencolor (title'color)
 print'at(24,1," hold <shift> key to stop the search")
 repeat
  searching:+1
  cursor(4,12)
  print using "###": searching
  read'record(searching)
  match'record
  if matching then
   clear'choices
   close'it
   current'show:=searching
   display(mark$<>" ")
   cursor(24,1)
   input "<return> = continue. a=abort: ": continue$
   clear'line(24)
   if continue$="" then
    open'it
    print'at(24,1,"hold <shift> key to stop search")
   endif
  endif
 until esc or shift or (searching=last'show) or (cont
 inue$>"") //wrap line
 close'it
 trap esc+
 if searching<>current'show then read'it(current'show)
endproc search
//
proc match'record
 matching:=false
 check(find'show'name$,show'name$)
 check(find'doctor$,doctor$)
 check(find'companion$,companion1$)
 check(find'companion$,companion2$)
 check(find'companion$,companion3$)
```

More ►

```
 check(find'adversary$,adversary1$)
 check(find'adversary$,adversary2$)
 check(find'location$,location$)
 check(find'id$,id$)
 check(find'mark$,mark$)
endproc match'record
//
proc check(find'text$,text$)
 if find'text$>"" and find'text$ in text$ then match
 ing:=true //wrap line
endproc check
//
proc clear'line(line)
 cursor(line,1)
 print tab(40)
endproc clear'line
//
proc browse
 print'at(24,1,"how many secs delay between shows:5 ")
 cursor(24,38)
 input "": delay
 format'screen
 pencolor (title'color)
 print'at(24,1," hold <shift> key to stop")
 trap esc- //disable stop key
 open'it
 repeat
  if current'show<last'show then current'show:+1
  read'record(current'show)
  display(mark$<>" ")
  pause(delay)
 until esc or shift or (current'show=last'show)
 trap esc+
 close'it
endproc browse
//
proc pause(seconds)
 now:=jiffies
 while jiffies<now+60*seconds do null
endproc pause
//
proc add'status
 status'outline
 pencolor (title'color)
 repeat
  cursor(23,2)
  input "what is your command: ": reply$
 until reply$ in "nyda"
 clear'status
 case reply$ of
 when "","y","Y"
  data'ok:=true
 when "n","N"
  data'ok:=false
 when "d","D"
  done'adding:=true
  data'ok:=true
 when "a","A"
  done'adding:=true
  data'ok:=false
 otherwise
  data'ok:=false // shouldnt be here
 endcase
endproc add'status
//
proc edit'status
 repeat
  clear'line(24)
  cursor(24,1)
  input "data ok? y=yes n=no a=abort: ": reply$
 until reply$ in "nya"
 clear'line(24)
 done'editing:=false
 case reply$ of
 when "","y","Y"
  data'ok:=true
 when "n","N"
  data'ok:=false
 when "a","A"
  done'editing:=true
  data'ok:=false
 otherwise
  data'ok:=false // shouldnt be here
 endcase
endproc edit'status
//
proc cursor(l,p) closed
 poke 214,l-1
 l:=1024+(l-1)*40
 poke 209,l mod 256
 poke 210,l div 256
 poke 211,p-1
endproc cursor
//
func jiffies closed
 return 256*256*peek(160)+256*peek(161)+peek(162)
endfunc jiffies
//
func file'exists(filename$) closed
 dim ds$ of 2
 open file 79,filename$,read
 ds$:=status$
 close file 79
 return ds$="00"
endfunc file'exists
//
proc print'at(r,c,text$)
 cursor(r,c)
 print text$
endproc print'at
//
proc create(name$,last'rec,rec'len) closed
 z:=zone
 zone 0
 open file 79,name$,random rec'len
 print file 79,last'rec: chr$(255),
 close file 79
 zone z
endproc create
//
proc page
 print chr$(147),
endproc page
//
proc display'bottom(show'data)
 pencolor (desc'color)
```

More ►

```
 print'at(14,1," id       :")
 print'at(15,1," mark     :")
 if show'data then
  pencolor (data'color)
  print'at(14,12,id$)
  print'at(15,12,mark$)
 endif
endproc display'bottom
//
proc status'outline
 cursor(16,1)
 pencolor (boxes)
 print "+---------+--------------------------+"
 for x:=1 to 5 do
  print "!         !                          !"
 endfor x
 print "+---------+--------------------------+"
 pencolor (title'color)
 print'at(17,2,"<return>")
 print'at(18,9,"y")
 print'at(19,9,"n")
 print'at(20,9,"d")
 print'at(21,9,"a")
 pencolor (desc'color)
 print'at(17,13,"default - same as y")
 print'at(18,13,"yes, data ok - do next one")
 print'at(19,13,"no, redo data input")
 print'at(20,13,"done - save and end input")
 print'at(21,13,"abort - stop - don't save")
endproc status'outline
//
proc clear'status
 for lines:=16 to 24 do
  clear'line(lines)
 endfor lines
endproc clear'status
//
proc clear'keys
 while key$>chr$(0) do null
endproc clear'keys
//
func value(s$) closed
 length:=len(s$)
 if length<1 or length>4 then return 0
 ones:=ord(s$(length))-ord("0")
 if ones<0 or ones>9 then return 0
 if length=1 then
  return ones
 else
  return ones+value(s$(1:length-1))*10 //recursive
 endif
endfunc value
//
func shift
 return peek(653)>0
endfunc shift
//
proc halt
 close
 page
 print "exiting to comal"+chr$(9)
 end
endproc halt
```

Statistics

By Bill Inhelder

That computers have radically changed people's lives has become an overworked cliche, yet an important aspect of that statement tends to be overlooked. Computers can be programmed to generate enormous amounts of data which require significant human effort and ability to summarize and interpret. The techniques of descriptive statistics provide the first step in the process of summarizing and interpreting data.

This statistical tutorial demonstration generates data involving computer and user interaction. It analyzes the data using the techniques of descriptive statistics.

Given a set of data, which single value best represents all of the data items? In statistics that value is designated as a measure of central tendency. There are three ways in which that value can be selected: mean, median and mode. The most familiar measure is the mean (in lay terms, the average) found by adding the items and dividing by the number of items. A serious problem with the mean is that it is sensitive to extreme values. For example, using the mean to determine *average* U.S. family income results in a value which is not representative because of the relatively few families with incomes of 6 digits or more.

For this reason federal and state agencies use the median (half the people have higher incomes and half have lower incomes) to report *average* U.S. family income. This statistic is far less sensitive to extreme values.

Finally there are occasions when the mode (the most frequently occurring value) is appropriate. This statistic is important

More ►

in the manufacture of shoes and clothing and the design of automobile interiors.

In the above example, the mean might be $37,000, the median about $30,000 and the mode $22,000. Even though the median is the favorite type of *average*, more information about the nature of the data is derived using all three. Since the mean is substantially larger than the median, we know that more than a few families are earning very large incomes. Yet the most typical family income is $22,000.

Measures of central tendency tell only part of the story. We need a statistic which tells us something about the degree to which the data is scattered from the mean. Are all the data items bunched up around the mean or are they widely dispersed? The statistic which helps to measure the degree of dispersion is the standard deviation. The larger the standard deviation the more disperse the data. Teachers generally prefer test scores with small standard deviations indicating that everyone understood the material to about the same degree. For many sets of data 2/3 of the items fall within 1 standard deviation of the mean.

An illustration will help focus on the concept of the standard deviation. Suppose two golfers, Sam and John, each drive 100 balls at a driving range. The results of the distances are recorded and the mean is calculated for each golfer. Both golfers have a mean of 150 yds. If Sam's distances give a standard deviation of 30 yds while John's distances amount to a standard deviation of 70 yds, who is more consistent? Assuming that about 2/3 of the data lie within 1 standard deviation of the mean, it implies that 2/3 of Sam's distances lie between 120 and 180 yds while 2/3 of John's distances lie between 80 and 220 yds. Certainly Sam is more consistent, while John is more erratic hitting some balls over 220 yds and slicing some balls less than 80 yds.

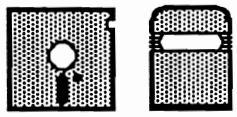
Usually raw data must be organized in order to analyse it further. If there are a large number of data items, tables become unwieldy. It becomes difficult to see the forest because of the trees. Grouping the data into defined intervals helps to see the big picture, albeit with less detail. Grouped data is good for graphic display in the form of bar graphs (technically, frequency histograms).

The program *statistical'demo* on *Today Disk #15* illustrates the above descriptive statistics using data provided by the user. The user's reaction time is measured in 30 trials. Depending upon alertness and degree of concentration, the timed values can be consistent or have a high degree of variability. Comparing the mean with the median will illustrate whether some extreme time values have been recorded. The larger the mean compared to the median, the greater the number of longer times in the data set. This might indicate a lack of concentration. A small standard deviation indicates alertness and consistency of response. This program offers optional screen or printer output of all data, descriptive statistics and bar graph.

The bar graphs of my reaction times tend to resemble a Poisson distribution (the first part of a roller coaster curve) rather than a normal distribution (the bell-shaped curve).

My apologies to the readers of *COMAL Today*, who are conversant with statistics, for the statistical oversimplifications in this article. ■

Recursive Designs



by D. Bruce Powell

There has been much interest recently in fractals. I translated 12 fractal designs from Logo into COMAL. I hope that some of my fellow COMALites will produce more fractal patterns.

Although I have not tried my hand at authoring a fractal design yet, the concept is not difficult; use recursion to repeat a basic pattern at various sizes, spacings, and rotations. I feel that there is a vast field of computer art awaiting our efforts!

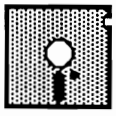
[Bruce was surprised to find out that many of his designs worked much faster in COMAL 0.14 than in COMAL 2.0 and asked us why. Part of the reason for this is that COMAL 0.14 uses keywords for graphics commands and COMAL 2.0 uses package commands. The graphics commands are faster in COMAL 2.0, but COMAL 2.0 also requires the USE command. Normally, USE does not slow down programs noticeably because it is only called once, but this program is not typical. Some procedures are called hundreds of times by recursion. Since the procedures are CLOSED, the USE graphics command is required to make the graphics commands available. Doing this several hundred times slows down the program.

Fortunately, there is a faster method. Once USE graphics has been issued at the beginning of the program, the commands forward, right, left, etc. are considered to be procedures by COMAL. They can be imported into other CLOSED procedures. This is much faster than the USE command.

The hooked snowflake design from Bruce's program is listed below to demonstrate the surprising difference in speed. Type it in, RUN it, get a cup of coffee, and watch it grow. When it finishes, comment out the USE commands in procedures rsnow and lsnow. Then remove the comments at the start of the IMPORT statement in the lines directly below the USE command. The next time you type in RUN, you better stick around to see what happens. The other 11 fractal designs are on Today Disk #15.]

```
USE graphics
USE system
hooksnow(175,3)
//
PROC start'drawing(x,y)
  graphicscreen(0)
  hideturtle
  background(14)
  pencolor(6)
  wrap
  moveto(x,y)
ENDPROC start'drawing
//
PROC hooksnow(size,level)
  start'drawing(210,16)
  rsnow(size,level)
ENDPROC hooksnow
//
PROC rsnow(size,level) CLOSED
  IMPORT lsnow,sine
  USE graphics
  // IMPORT right,left,forward
  IF level<>0 THEN
    unit':=size/3
    sunit:=2*size*sine(60)/9
    left(60)
    lsnow(unit',level-1); rsnow(unit',level-1)
    right(60); rsnow(unit',level-1)
    right(60); rsnow(unit',level-1)
    right(150)
    rsnow(sunit,level-1); lsnow(sunit,level-1)
    left(60); rsnow(sunit,level-1)
    left(60); rsnow(sunit,level-1)
    rsnow(sunit,level-1)
    left(90)
    lsnow(unit',level-1); rsnow(unit',level-1)
  ELSE
    forward(size)
  ENDIF
ENDPROC rsnow
//
PROC lsnow(size,level) CLOSED
  IMPORT rsnow,sine
  USE graphics
  // IMPORT right,left,forward
  IF level<>0 THEN
    unit':=size/3
    sunit:=2*size*sine(60)/9
    lsnow(unit',level-1); rsnow(unit',level-1)
    right(90)
    lsnow(sunit,level-1); rsnow(sunit,level-1)
    right(60)
    lsnow(sunit,level-1); right(60)
    rsnow(sunit,level-1); lsnow(sunit,level-1)
    left(150)
    lsnow(unit',level-1); left(60)
    lsnow(unit',level-1); left(60)
    lsnow(unit',level-1); rsnow(unit',level-1)
    right(60)
  ELSE
    forward(size)
  ENDIF
ENDPROC lsnow
//
FUNC sine(angle)
  RETURN SIN(PI*angle/180)
ENDFUNC sine
```

Introduction to Procedures



by David Stidolph

The process of writing a program can be described as defining the problem and breaking it into small, manageable, separate sections. With this method, a complex program can be designed as a series of small sections, called procedures and functions.

A procedure is a block of program statements, called by name, which perform a useful task. The procedure can be called from anywhere in the program. This saves memory because it is only defined once, not every time it is needed. Another advantage of procedures is that once the program has been **RUN**, the procedure may be called from direct mode. This powerful feature lets you add command words to COMAL (while the procedure is in memory). The following example shows a very simple procedure definition and how it is called.

```
0010 print "I am a regular statement."
0020 print'procedure
0030 print "This is after the call."
0040 end
0050 //
0060 proc print'procedure
0070   print "Inside the procedure."
0080 endproc print'procedure
```

When **RUN**, the program prints:

I am a regular statement.
Inside the procedure.
This is after the call.

To see how **print'procedure** has become a command, call it from the immediate mode. Just type:

```
print'procedure
```

The response is:

Inside the procedure.

Though this procedure is simple, it shows how to define and call a procedure. Line 60 is the procedure header. It defines the name of the procedure, and what information, if any, needs to be passed to it (more on this later). Line 80 marks the end of the procedure. When this statement is reached, the procedure is finished and control returns to the line following the procedure call. Line 20 shows how to call a procedure - by using its name. In this case the procedure is called **print'procedure**. Notice that the sample output shows how control passes to the procedure and back to the statement following the call.

But why go through all the trouble and typing just to print three lines? Procedures are most useful when called from several points in a program. The following procedure will clear the keyboard buffer. This may not seem very important, but many programs spend time drawing graphics, doing file access, or computing some formula. During this time the user may press a key on the keyboard by mistake. It is best to clear the keyboard buffer before you do any **INPUT**.

```
9000 proc clearkeys
9010   while key$>chr$(0) do null
9020 endproc clearkeys
```

As important as procedures are, typing each procedure I needed into every program would require more time than I spend designing the entire program. COMAL gives me the ability to maintain a procedure library. I **LIST** my procedures to disk and **ENTER** them into programs as I need them - without retyping the procedure.

More ►

Notice that I numbered the lines of this procedure starting at 9000. There is a reason for this. In COMAL 0.14, lines merged from disk overwrite identically numbered lines in memory. If I numbered **clearkeys** to start at line 10 as in the previous example, it would replace lines 10-30 of the program it was entered into. As a rule, number the procedure to start at line 9000 before listing it to disk for your procedure library.

To list **clearkeys** to disk, type it in and use the following command:

```
list "clearkeys.proc"
```

COMAL will now list the program lines in memory into a file called *clearkeys.proc*. I put the *.proc* after the name *clearkeys* so that I will remember this is a listed procedure. When I want to merge this procedure with another program I have in memory, I will use the commands:

```
enter "clearkeys.proc"
renum
```

The **RENUM** command forces line numbers to start at 10 and then increment in units of 10. After the **RENUM** you may merge in a second procedure that also is numbered from 9000 without any conflicts. A variation of the **RENUM** command can make line numbers start at 9000 to let you list a procedure to disk:

```
renum 9000
```

I strongly advise you to always break your program into small procedures. It makes it easier for you to debug your program, and makes your program more readable. Make sure you use meaningful names for your procedures. **Convert'dollars'to'yen** is easier to understand than **cvtdy**. Since COMAL puts all variable names into a table and tokenizes them within the program, virtually no extra space is wasted by long names. I also suggest you keep your procedures short enough to see the entire procedure on the screen at once.

FUNCTIONS

Functions are like procedures except they return values. A function call looks like a variable, but acts like a constant. For example, it can appear on the right side of an assignment statement (:=), but not on the left. Function values may be printed or used in tests for the **IF**, **CASE**, **WHILE**, and **UNTIL** statements. A function call may not stand alone on a line like procedure calls can, because the value of the function would have nowhere to go.

You have been using functions in your programs already if you have used **ABS**, **COS**, **RND**, **PEEK**, or any other built in function. COMAL 0.14 lacks the built in function of **PI**, so we will define it here:

```
func pi
  return 3.14159266
endfunc pi
```

Notice that I did not print the line numbers. From here on I will just print the statements - you can provide line numbers with the **AUTO** command. When called, **pi** returns the value 3.14159266. The following program uses **pi** to calculate the area of a circle. (If you are typing this in from COMAL 2.0 where **PI** is a built in function, you do not need to include the last four lines.)

```
input "Enter radius of circle: ": radius
print "Area of the circle is";
print pi*radius^2
//
```

More ►

```
func pi
  return 3.14159266
endfunc pi
```

PARAMETERS

Parameters are values or variables you pass to the procedure or functions. For example, **PRINT RND(1,10)** will print a random number between 1 and 10. In this case the numbers 1 and 10 are the parameters for the **RND** function.

The program above figures the area of a circle. The function below will return the area of a circle from a given radius:

```
func circle'area(radius)
  return pi*radius^2
endfunc circle'area
```

Radius is the parameter of the function **circle'area**. When you call **circle'area** you must put a number, or numeric variable, inside of parentheses right after the name. The following is the find area program rewritten to use the circle'area function.

```
input "Enter radius of circle: ": value
print "Area of the circle is";
print circle'area(value)
//
func pi // don't type this function in
  return 3.14159266 // for COMAL 2.0
endfunc pi
//
func circle'area(radius)
  return pi*radius^2
endfunc circle'area
```

I changed the name of **radius** in the program to **value** to show you that the variables you pass to procedures or functions don't need to have the same name. The parameter name in the procedure or function header is just an easy way to refer to the value being passed. If you do happen to use the same name in the procedure header, there is no problem. COMAL will automatically create a new variable using the name in the header and use that definition only until the procedure or function ends.

Parameters can be of any standard variable type (real, integer, string, or array). The main purpose of parameters is to make procedures and functions more versatile. For example, a poor way to write the program above could be:

```
input "Enter radius of circle: ": value
print "Area of the circle is";
print circle'area
//
func pi // don't type this function in
  return 3.14159266 // for COMAL 2.0
endfunc pi
//
func circle'area
  return pi*value^2
endfunc circle'area
```

In this case, there is no parameter; this function is specific to this program. You always have to set the variable **value** to the radius of the circle before calling function **circle'area**. This is far less transportable than the original.

REFERENCE PARAMETERS

Suppose we make a general purpose procedure to swap the values of two variables.

```
proc swap(first,second) closed
  temp:=first
  first:=second
  second:=temp
endproc swap
```

More ►

Here is some code to test the procedure:

```
a:=1; b:=2
swap(a,b)
print "a=",a
print "b=",b
```

When **RUN**, you find that after **swap**, **a** is still **1** and **b** is still **2**. The problem is that when **swap** was called, it created the variables **first** and **second**. When the procedure ended, their values disappeared. Variables **a** and **b** were never changed. Reference parameters must be used in the **swap** procedure. If changes are made to a reference parameter, the changes are also made to the variable that it represents (even if they have different names). Here is the correct procedure:

```
proc swap(ref first,ref second) closed
  temp:=first
  first:=second
  second:=temp
endproc swap
```

Now if you use the same calling code, variable **a** will equal **2** and variable **b** will equal **1** after calling **swap**. You should make parameters referenced if you want to change their values within the procedure. One word of caution: only use the reference notation when you need it. If you make a parameter referenced, you will find you can't pass it a constant (or function value). For example, the following command would give you an error with the swap procedure above.

```
swap(5,9)
```

BUILDING BLOCKS

Procedures and functions are the building blocks of a program. They allow you to *borrow* code from other programs (so you don't have to write it yourself) and to quickly assemble your program from blocks of code that already work. This means you can concentrate on the logic of your program and how it works - not on whether there is a problem positioning the cursor.

The following pages list many commonly used COMAL 0.14 procedures and functions that have been written in COMAL (and sent to COMAL Users Group) since the start of *COMAL Today*. They all are on *Today Disk #15*. To make your own library disk of procedures and functions, follow these steps for **each** one:

1) Type: NEW
2) Type: AUTO 9000
3) Type in the procedure or function
4) COMAL 0.14: Hit <return> to stop AUTO
5) Store to disk: LIST "NAME.PROC"

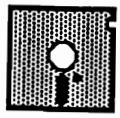
SPECIAL NOTES:

There are many more procedures and functions available for your library disk. Each issue of *COMAL Today* has provided new routines (also available on each *Today Disk*). We are including the **1520 Plotter** routines on *Today Disk #15* for your use. They are explained in *COMAL Today #7* on page 62. Another important set of routines are the **Disk Get** routines, originally from the *COMAL Handbook* Appendix D. They also are on *Today Disk #2.*

These routines are meant for COMAL 0.14 users. Some may be used by COMAL 2.0 users. However, use the **MERGE** command rather than **ENTER**, and it is not necessary to issue a **RENUM** command after it.

Finally, you can use the **listerine** program from *COMAL Today #14* to pull out procedures and functions from any of your programs. ■

Procedures & Functions Our Collection



Procedure / Function Index

FREEFILE

This function returns a number that can safely be used as a new file number. It is a built-in keyword in IBM COMAL.

```
func freefile closed
  fnum:=20
  repeat
    fnum:-1; found:=true
    for x:=601 to 600+peek(152) do
      if peek(x)=fnum then found:=false
    endfor x
  until found
  return fnum
endfunc freefile
```

INPUT'AT

This procedure positions the cursor to the given location and does an input using **text$** as the prompt, and **variable$** as the string to assign the input to.

```
proc input'at(row,col,text$,ref variable$) closed
  addr:=1024+(row-1)*40
  poke 209,addr mod 256
  poke 210,addr div 256
  poke 211,col-1
  poke 214,row-1
  input text$: variable$
endproc input'at
```

PRINT'AT

This procedure moves the cursor to the appropriate postion on the screen and prints the text you pass it. **Row** should be numbered between 1 and 25, and **col** should be numbered between 1 and 40. Note the comma after **text$** in the **PRINT** statement. This means the cursor will be left at the position immediately following the printed string.

```
proc print'at(row,col,text$) closed
  addr:=1024+(row-1)*40
  poke 209,addr mod 256
  poke 210,addr div 256
  poke 211,col-1
  poke 214,row-1
  print text$,
endproc print'at
```

SHIFT'WAIT

This procedure will print the flashing message *press shift* on the screen until the shift key is pressed. The message is erased before the procedure exits.

```
proc shift'wait closed
  shift'flag:=653
  while not peek(shift'flag) and not esc do
    print "press shift"+chr$(145)
    print "           "+chr$(145)
  endwhile
endproc shift'wait
```

More ►

PAGE

This command emulates the **PAGE** command of COMAL 2.0. If the output device is the screen, a clearscreen character (147) is printed, otherwise the printer is assumed and a form feed character (12) is printed. Some printers do not have a form feed capability and ignore this command.

```
proc page closed
  if peek(152)=0 then
    print chr$(147),
  else
    print chr$(12),
  endif
endproc page
```

CURSOR

This procedure moves the cursor to the given position. Note that position (1,1) is the top left hand corner.

```
proc cursor(row,col) closed
  poke 211,col-1
  poke 209,(1024+(row-1)*40) mod 256
  poke 210,(1024+(row-1)*40) div 256
  poke 214,row-1
endproc cursor
```

CURCOL

This function returns the column postion of the cursor (1-40).

```
func curcol
  return peek(211)+1
endfunc curcol
```

CURROW

This function returns the row position of the cursor (1-25). Row 1 is the top line of the screen.

```
func currow
  return peek(214)+1
endfunc currow
```

FILLKEYS

Here is a procedure that will fill the keyboard buffer with the string you pass to it. Remember that the buffer can only hold 10 keystrokes.

```
proc fillkeys(string$) closed
  if len(string$)>10 then string$:=string$(1:10)
  for x:=1 to len(string$) do
    poke 630+x,ord(string$(x))
  endfor x
  poke 198,len(string$)
endproc fillkeys
```

CLEARKEYS

This procedure will clear the keyboard buffer.

```
proc clearkeys
  while key$ > chr$(0) do null
endproc clearkeys
```

INKEY

This procedure will turn the blinking cursor on and wait for a key to be pressed. The referenced string passed to this procedure is assigned that new key value and the cursor is turned off. No character is printed by this procedure.

```
proc inkey(ref c$) closed
  c$:=key$
  if c$=chr$(0) then
    poke 204,0 // cursor on
    repeat
      c$:=key$
    until c$<>chr$(0)
    if peek(207) then poke 205,1
    while peek(207) do null // is on?
    poke 204,1 // cursor off
  endif
endproc inkey
```

More ►

REPEATKEYS

This procedure can turn on, or off, the auto repeat key feature of the C64. By default this is off, but it can be turned on. With this feature, any key will start repeating while you hold it down (after an initial half second wait).

```
proc repeatkeys(state)
  if state then
    poke 650,128
  else
    poke 650,0
  endif
endproc repeatkeys
```

SOUND CONTROLS

These procedures emulate the commands in the COMAL 2.0 sound package.

```
proc setfrequency(v,f)
  poke 54272+(v-1)*7,f mod 256
  poke 54272+(v-1)*7+1,f div 256
endproc setfrequency
//
proc pulse(v,p)
  poke 54272+(v-1)*7+2,p mod 256
  poke 54272+(v-1)*7+3,p div 256
endproc pulse
//
proc adsr(v,a,d,s,r)
  poke 54272+(v-1)*7+5,a*16+d
  poke 54272+(v-1)*7+6,s*16+r
endproc adsr
//
proc gate(v,on'off) closed
  temp:=int(peek(1020+v)/2)*2
  if on'off then temp:+1
  poke 1020+v,temp
  poke 54272+(v-1)*7+4,temp
endproc gate
//
proc filtertype(lo,band,hi,v3) closed
  temp:=peek(1019) mod 16
  temp:+v3*128+hi*64+band*32+lo*16
  poke 1019,temp
  poke 54272+24,temp
endproc filtertype
//
proc filterfreq(f)
  poke 54272+21,f mod 8
  poke 54272+22,f div 8
endproc filterfreq
//
proc filter(v1,v2,v3,e) closed
  temp:=peek(1018)
  temp:=(temp div 16)*16
  temp:+v1+v2*2+v3*4+e*8
  poke 54272+23,temp
  poke 1018,temp
endproc filter
//
proc sync(v,s) closed
  temp:=peek(1020+v)
  temp:=(temp div 4)*4+(temp mod 2)
  temp:+s*2
  poke 1020+v,temp
  poke 54272+(v-1)*7+4,temp
endproc sync
//
proc ringmod(v,r) closed
  temp:=peek(1020+v)
  temp:=(temp div 8)*8+(temp mod 4)
  temp:+r*4
  poke 1020+v,temp
  poke 54272+(v-1)*7+4,temp
endproc ringmod
//
proc soundtype(v,t) closed
  temp:=peek(1020+v)
  temp:=temp mod 16
  if t>0 then temp:+(2t-1))*16
  poke 1020+v,temp
  poke 54272+(v-1)*7+4,temp
endproc soundtype
//
proc resonance(r) closed
  temp:=peek(1018)
  temp:=temp mod 16
  temp:+r*16
  poke 54272+23,temp
  poke 1018,temp
endproc resonance
//
proc volume(v) closed
  temp:=peek(1019)
  temp:=(temp div 16)*16+v
  poke 54272+24,temp
  poke 1019,temp
endproc volume
```

GETBACKGROUND

This function returns the current background color of the text or graphics screen.

```
func getbackground closed
  return peek(53281) mod 16
endfunc getbackground
```

More ►

PLOT'TEXT

This procedure will plot lower/upper case on the graphic screen. Instead of using the standard **PLOTTEXT** locations, the same row/column positions from the text screen are used (1,1 is in the upper left hand corner). The machine code is used to translate the characters into a format usable by the **PLOTTEXT** command.

```
proc plot'text(row,col,c$) closed
  start:=828
  if peek(start)<>160 then
    poke 29260,216 // lower'case
    poke start,160 // relocatable
    poke start+1,3 // code
    poke start+2,177
    poke start+3,51
    poke start+4,170
    poke start+5,200
    poke start+6,177
    poke start+7,51
    poke start+8,201
    poke start+9,192
    poke start+10,144
    poke start+11,4
    poke start+12,233
    poke start+13,96
    poke start+14,145
    poke start+15,51
    poke start+16,202
    poke start+17,208
    poke start+18,242
    poke start+19,96
  endif
  if c$<>"" then sys start
  plottext (col-1)*8,192-(row-1)*8,c$
endproc plot'text
```

TEXTCOLORS

This procedure emulates the **textcolors** command of COMAL 2.0. If you wish to leave one of the settings alone, use -1 as the parameter. For example, to set the text color to red and leave background and border colors alone: **textcolors(-1,-1,2)**

```
proc textcolors(bor,ba,text)
  if bor>=0 then border bor
  if ba>=0 then background ba
  if text>=0 then pencolor text
endproc textcolors
```

LOADFONT

This procedure will load a character font into memory.

```
proc loadfont(name$)
  if peek(850)<>169 then
    checksum:=0
    for i:=850 to 881 do
      read num
      poke i,num
      checksum:+num
    endfor i
    if checksum<>3972 then
      print "data error!"
      stop
    endif
  endif
  data 169,8,170,160,0,32,186,255
  data 169,16,162,60,160,3,32,189
  data 255,169,0,162,0,160,200,32
  data 213,255,169,8,32,195,255,96
  for i:=1 to len(name$) do
    poke 827+i,ord(name$(i))
  endfor i
  poke 859,i-1
  sys 850
endproc loadfont
```

SAVESHAPE

This procedure emulates the COMAL 2.0 command **SAVESHAPE**. It will save a sprite definition area (0-63) to disk in the standard format used by COMAL 2.0.

```
proc saveshape(spr,filename$) closed
  z:=zone // original zone setting
  zone 0 // no space for zones
  if spr>=0 and spr<64 then
    dim ds$ of 2
    open file 81,filename$,write
    ds$:=status$
    if ds$="00" then
      block:=49152+spr*64
      for i#:=0 to 63 do
        byte:=peek(block+i#)
        print file 81: chr$(byte),
      endfor i#
    endif
    close file 81
  endif
  zone z // return zone to original
endproc saveshape
```

More ►

LOADSHAPE

This procedure emulates the **LOADSHAPE** command in COMAL 2.0. It will load a sprite shape (64 bytes) in to a sprite definition area (0-63).

```
proc loadshape(spr,filename$) closed
  dim ds$ of 2
  open file 81,filename$,read
  ds$:=status$
  if ds$="00" and spr>=0 and spr<64 then
    a:=828
    poke a+0,162 //   ldx #81
    poke a+1,81
    poke a+2,32 //    jsr $ffc6
    poke a+3,198
    poke a+4,255
    poke a+5,160 //   ldy #00
    poke a+6,0
    poke a+7,32 // lp  jsr $ffcf
    poke a+8,207
    poke a+9,255
    poke a+10,153 //  sta spr,y
    poke a+11,(49152+spr*64) mod 256
    poke a+12,(49152+spr*64) div 256
    poke a+13,200 //  iny
    poke a+14,192 //  cpy #64
    poke a+15,64
    poke a+16,208 //  bne lp
    poke a+17,245
    poke a+18,96 //   rts
    sys a
  endif
  close file 81
endproc loadshape
```

CIRCLE

This procedure will draw a true circle on the graphics screen. It is among the fastest written for COMAL 0.14.

```
proc circle(x,y,r) closed
  y':=0
  penup
  for i#:=1 to 64 do
    t:=r*.995004165-y'*.0998334166
    y':=y'*.995004165+r*.0998334166
    r:=t
    sx:=1.4*r+x
    sy:=y-y'
    drawto sx,sy
    pendown
  endfor i#
endproc circle
```

MOUNT

This procedure initializes the disk drive.

```
proc mount closed
  pass "i0"
endproc mount
```

FILE'EXISTS

This 0.14 function returns **TRUE** (1) if the filename passed to it exists on the current disk in the disk drive. If the file does not exists, or if there is <u>any</u> disk problem, the function will return **FALSE** (0). It is a good idea to always check to make sure that any file you need to read from is on the disk in the disk drive before you open the file, so that you can handle the problem - rather than have your program crash.

```
func file'exists(filename$) closed // 0.14 only
  dim d$ of 2
  open file 79,filename$,read
  d$:=status$
  close file 79
  return d$="00"
endfunc file'exists
```

CREATE

This procedure will create a **RANDOM** file on disk. You can specify the name of the file, the size of each record (1-254) and how many records (1 to the limit of disk space) it can hold. Remember that when you **WRITE** to disk, strings take up their length plus two bytes (tells how long the string is), and both real and integer variables take up 5 bytes.

```
proc create(name$,last'rec,rec'len) closed
  z:=zone
  zone 0
  open file 79,name$,random rec'len
  print file 79,last'rec: chr$(255),
  close file 79
  zone z
endproc create
```

More ►

LOAD'OBJ

This procedure loads machine code from disk back into its original location.

```
proc load'obj(name$) closed
  poke 157,0
  poke 858,169
  poke 859,0
  poke 860,76
  poke 861,213
  poke 862,255
  open file 78,name$+",p",read
  sys 858
  close file 78
endproc load'obj
```

SAVE'OBJ

This procedure saves a section of memory to disk. The filename, starting address, and ending address must be passed to the procedure. It will only save the *top* level of RAM so this procedure could not be used to save a high-res screen.

```
proc save'obj(name$,start'addr,end'addr) closed
  open file 78,name$+",p",write
  print file 78: chr$(start'addr mod 256),
  print file 78: chr$(start'addr div 256),
  for addr:=start'addr to end'addr do
    print file 78: chr$(peek(addr)),
  endfor addr
  close file 78
endproc save'obj
```

DRIVE8

This procedure will change a unit 9 drive to a unit 8 drive (as long as power to the drive stays on and it is not reset).

```
proc drive8 closed
  open file 15,"",unit 9,15,read
  dim s$ of 10
  s$:="m-w"+chr$(119)+chr$(0)+chr$(2)
  print file 15: s$+"(h",
  close file 15
endproc drive8
```

DRIVE9

This procedure will change a disk drive set for unit number 8 to unit number 9.

```
proc drive9 closed
  open file 15,"",unit 8,15,read
  dim s$ of 10
  s$:="m-w"+chr$(119)+chr$(0)+chr$(2)
  print file 15: s$+")i",
  close file 15
endproc drive9
```

1571 DISK PROCEDURES

The following four procedures can only be used with a Commodore 1571 disk drive. **Front'side** sets the 1571 disk drive to use the front side of a 1571 disk. **Back'side** sets the 1571 disk drive to use the back side of a 1571 formatted disk as if it were a separate disk than the front side. It has its own directory and files. This procedure cannot be used to read double sided 1541 formatted disks such as the *Today Disks* because of the opposite directions that the disks spin. **Double'sided** sets the 1571 disk drive to use both sides of a 1571 formatted disk. There is one directory for both sides, but you get double the blocks free compared to a 1541 formatted disk. **Single'sided** sets the 1571 disk drive to function as a 1541 disk drive.

```
proc double'sided  // 1571 mode
  pass "u0>m1"
endproc double'sided
//
proc single'sided  // 1541 mode
  pass "u0>m0"
endproc single'sided
//
proc front'side  // while in 1541 mode
  pass "u0>h1"
endproc front'side
//
proc back'side  // while in 1541 mode
  pass "u0>h0"
endproc back'side
```

More ►

READING DISK BLOCKS

The disk'get procedure has been written and re-written for over 2 years. In almost every case it has been used to read information from specific disk blocks. The following routines can be used to read a specific block on a disk into a 256 byte string. The first function is the one you call, and will return **TRUE** if any disk error was encountered.

```
func read'block(track,sector,ref block$) closed
  dim t$ of 2, s$ of 2, ds$ of 2
  mem:=1000
  str(track,t$); str(sector,s$)
  if peek(mem)<>162 then disk'get'init(mem)
  open file 2,"#2",unit 8,2,read
  ds$:=status$
  if ds$="00" then
    pass "u1: 2 0 "+t$+" "+s$
    block$(1:258):=""
    if block$="" then null
    start:=peek(51)+peek(52)*256+4
    poke 51,start mod 256
    poke 52,start div 256
    sys 1000
  endif
  close file 2
  return ds$<>"00"
endfunc read'block
//
proc disk'get'init(start) closed
  a:=start
  while not eod do
    read byte
    poke a,byte
    a:+1
  endwhile
  data 162,2,32,198,255,160,0
  data 32,207,255,145,51,200,208
  data 248,32,204,255,96
endproc disk'get'init
//
proc str(num,ref string$)
  string$:=chr$((num mod 10)+48)
  if num div 10 then
    string$:=chr$((num div 10)+48)+string$
  endif
endproc str
```

DIR

COMAL 2.0 has the command **DIR** which can used from within a running program to show a directory of any connected disk drive. The following command modifies the built in **CAT** command of COMAL 0.14 to achieve the same results. The first parameter is the name, if any, you wish to search for in the directory. If you want all names, use "*". The second parameter is the device number (normally 8 or 9).

```
proc dir(name$,device) closed
  trap esc-
  for x:=1 to len(name$) do
    poke 834+x,ord(name$(x))
  endfor x
  poke 26997,x
  poke 27013,device
  poke 27110,96
  sys 26996
  poke 27110,76
  poke 26997,1
  poke 27013,8
  while esc do null
  trap esc+
endproc dir
```

GETTIME

This function returns the current number of *jiffies* (1/60th of a second) since the computer was turned on, or the clock was reset with **SETTIME.**

```
func gettime closed
 return 256*256*peek(160)+256*peek(161)+peek(162)
endfunc gettime
```

SETTIME

This procedure sets the jiffy clock. It's only parameter is the number of jiffies (1/60th of a second) to set the clock to.

```
proc settime (jiffies) closed
  poke 162,jiffies mod 256
  poke 161,jiffies div 256 mod 256
  poke 160,jiffies div 65536
endproc settime
```

More ►

JOYSTICK

This procedure emulates the **JOYSTICK** command in the COMAL 2.0 cartridge. The first parameter specifies which port to check (one or two). The next two parameters are referenced and will be set to the direction the joystick returns and the status of the fire button. **Direction** will be set to zero if the joystick is not moved, otherwise it will be set to one if forward, two if to the upper right, etc, clockwise through eight for upper left.

```
proc joystick(port,ref direction,ref button) closed
  if port=1 then
    mem:=peek(56321)
  else
    mem:=peek(56320)
  endif
  button:=1-((mem mod 32) div 16)
  case 15-(mem mod 16) of
  when 1
    direction:=1
  when 2
    direction:=5
  when 4
    direction:=7
  when 5
    direction:=8
  when 6
    direction:=6
  when 8
    direction:=3
  when 9
    direction:=2
  when 10
    direction:=4
  otherwise
    direction:=0
  endcase
endproc joystick
```

```
    1
  8   2
7   0   3
  6   4
    5
```

LIGHTPEN

This procedure will scan the lightpen registers until a position within the screen boundry is detected. The lightpen must be plugged into port #1. The referenced variables are assigned the x-y coordinates.

```
proc lightpen(ref x,ref y) closed
  repeat
    x:=peek(53267)*2-72
    y:=246-peek(53268)
  until x<=320 and x>=0 and y<=199 and y>=0
endproc lightpen
```

KOALA

This procedure allows you to read the status of a *Koala Pad* or *Animation Station*. You specify which port it is plugged into, and the other parameters are changed to reflect its state.

```
proc koala(port,ref x,ref y,ref lb,ref rb) closed
  j:=15-peek(3-port+56319) mod 16
  lb:=((j mod 8) div 4)>0 // left button
  rb:=(j div 8)>0 // right button
  poke 56333,1
  d:=peek(56322)
  poke 56322,192
  poke 56320,64*port
  x:=peek(54297)
  y:=255-peek(54298)
  poke 56322,d
  poke 56333,129
endproc koala
```

PADDLE

This procedure emulates the **PADDLE** command of the COMAL 2.0 cartridge. The first parameter specifies which port to check, and the rest of the parameters will be set according to the paddles connected to that port. If no paddles are connected, then the x and y variables will be set to 255 and the two fire buttons will be zero.

```
proc paddle(port,ref x,ref y,ref button1,ref button2) closed
  cia:=56320; sid:=54272
  poke cia+13,1 //disable timer a interrupt
  ddra:=peek(cia+2)
  poke cia+2,192
  poke cia,64*port
  x:=peek(sid+25); y:=peek(sid+26)
  poke cia+2,ddra
  poke cia+13,129 //enable timer
  mem:=peek(cia+2-pair)
  button1:=1-((mem mod 16) div 8)
  button2:=1-((mem mod 8) div 4)
endproc paddle
```

More ►

PI

This function returns the value of **PI** (a built in function of COMAL 2.0).

```
func pi
  return 3.14159265
endfunc pi
```

EXPAND'RAM

This procedure will expand the total amount of programming space from 9,902 bytes to 11,838 bytes. The new memory space cannot be used until a NEW command has been issued.

```
proc expand'ram closed
  if peek(18728)<>183 then
    for addr:=45969 to 45981 do
      poke addr+1024,peek(addr)
    endfor addr
    poke 4569,183
    poke 15256,183
    poke 17458,183
    poke 18728,183
    poke 21833,183
    poke 24332,183
    poke 24606,183
    poke 26866,183
    poke 28711,183
    poke 28727,183
    poke 29977,183
    poke 34441,183
    poke 751,2
    poke 2066,144
    poke 2067,183
  endif
endproc expand'ram
```

LOAD'ERRORS

This procedure will load the error file *comalerrors* into the hidden memory underneath the I/O block at $d000 and *patch* COMAL to look there for the error messages. With this procedure you can have error messages in memory, but still have the entire sprite area at $c000 free for sprites or other machine code.

```
proc load'errors closed
  checksum:=0
  for address:=49152 to 49200 do
    read value
    poke address,value
    checksum:+value
  endfor address
  if checksum<>6637 then
    print "load'errors code is incorrect"
    stop
  endif
  open file 8,"comalerrors",read
  ds$:=status$
  if ds$="00" then sys 49152
  close file 8
  data 162,8,32,198,255,169,208,141
  data 32,192,160,0,32,207,255,170
  data 32,183,255,208,24,120,165,1
  data 72,41,240,133,1,138,153,0,208
  data 104,133,1,88,200,208,228,238
  data 32,192,208,223,32,204,255,96
  checksum:=0
  for loc:=6482 to 6497 do
    read byte
    checksum:+byte
    poke loc,byte
  endfor loc
  for loc:=47006 to 47049 do
    read byte
    checksum:+byte
    poke loc,byte
  endfor loc
  for loc:=1 to 6 do
    read address
    poke address+1,158
    poke address+2,183
    checksum:+address
  endfor loc
  if checksum<>47040 then
    print "checksum error for data statements"
    print " *** please reload comal ***"
    stop
  endif
  data 169,0,141,171,183
  data 169,208,141,172,183
  data 169,0,133,144,240,12
  data 165,144,208,39,120
  data 165,1,72,41,248
  data 133,1,174,0,208
  data 238,171,183,208,17
  data 238,172,183,169,216
  data 205,172,183,208,7
  data 169,64,133,144,56
  data 176,1,24,104,133
  data 1,138,88,96
  data 6510,6521,6529
  data 6538,6543,6548
endproc load'errors
```

More ►

BITWISE OPERATIONS

The following three functions provide the basic BIT operations found in COMAL 2.0 (on a single byte basis only). All three call a fourth function that does machine code initialization and calling.

```
func bitand(n,m)
 return bit'ml(n,m,41)
endfunc bitand
//
func bitor(n,m)
 return bit'ml(n,m,9)
endfunc bitor
//
func bitxor(n,m)
 return bit'ml(n,m,73)
endfunc bitxor
//
func bit'ml(n,m,o)
 poke 2045,169
 poke 2046,n
 poke 2047,o
 poke 2048,m
 poke 2049,141
 poke 2050,232
 poke 2051,7
 poke 2052,96
 sys 2045
 return peek(2024)
endfunc bit'ml
```

ROUND

This function will return the value of any number rounded to the nearest integer.

```
func round(number) closed
  return sgn(number)*int(abs(number)+.5)
endfunc round
```

TRUNC

This function will return the integer portion any number. It differs from the INT function only on negative numbers.

```
func trunc(number) closed
  return sgn(number)*int(abs(number))
endfunc trunc
```

VAL

This function returns the numeric equivalent of the number stored in the string passed to it. It uses the disk drive to do the conversion, but does not require a disk in the drive and will not read or write from a disk in any case. Warning: an error will result if the string cannot be evaluated to a number.

```
func val(x$) closed
  open file 100,"#",unit 8,2,read
  print file 100: x$
  pass "b-p:2,1"
  input file 100: y
  close file 100
  return y
endfunc val
```

STR

This procedure will take the number you pass it in the first parameter and set the referenced string to the ASCII equivalent of that number. It uses the disk drive for the transformation, but does not require a disk in the drive.

```
proc str(x,ref y$) closed
  open file 100,"#",unit 8,2,read
  print file 100: x
  pass "b-p:2,1"
  input file 100: y$
  close file 100
endproc str
```

ENHANCED

This function returns **TRUE** if *The Enhancer, Today Disk #12* is in effect. Certain programs which require space at $c000 should use this function to make sure that *The Enhancer* is not in use.

```
func enhanced
  return peek(806)+peek(807)*256=49920
endfunc enhanced
```

More ►

CALL

This procedure works in a manner similar to **CHAIN**, except that it will **NEW** the current program in memory before chaining the other program. The procedure works by clearing the screen, printing the appropriate commands, and putting carriage returns in the keyboard buffer.

```
proc call(filename$) closed
  print chr$(147),"new"
  print chr$(17),"chain""",filename$,""""
  poke 631,19
  for x:=632 to 634 do poke x,13
  poke 198,4
  end
endproc call
```

TURBO

This procedure can only be used with the C128 computer. It will set the FAST mode (double speed with screen blanked) according to the parameter, **state**. If **state** is zero (**FALSE**) then FAST mode is disabled, otherwise it is turned on.

```
proc turbo(state)
  if state then
    poke 53296,3
  else
    poke 53296,0
  endif
endproc turbo
```

FREE

This function will return the amount of available memory. Similar to **SIZE**, but can be called from within a running program. Use it to make sure memory has been expanded, or to set the size of arrays.

```
func free closed
  bottom:=peek(65)*256+peek(64)
  top:=peek(67)*256+peek(66)
  return top-bottom
endfunc free
```

Merger

by David Stidolph

Merge'procs on *Today Disk #15* is a utility program that will merge procedures and funcctions from separate files on a disk into one mergeable file. It reads the directory and determines which files you can choose from (those SEQ files with a "." period in their name, such as *pause.proc* or *func.freefile*). It then uses an excellent text file window routine from David Warman to scroll the filenames for you to choose from. *[The scrolling window technique is also used in the menu program on Today Disk #15.]* To include a file, hit *<return>* when a filename is by the pointer.

After you mark all the files you want put together, the program asks for the filename to store them into. It then reads the appropriate files, automatically numbers the lines, and stores them together into the specified file.

PENCOLOR FLIP

Now you can plot or draw something on the 2.0 graphics screen and then remove it, properly replacing all the bits as they originally were. To enable **flip** mode, issue this POKE:

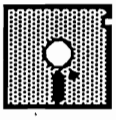
POKE $c462,$5d

To restore things to normal, enter this:

POKE $c462,$1d

Next issue we plan to show some interesting uses of this. If you want to contribute, send your ideas right away.

Program Construction



by Captain COMAL

The preceding pages provide you with many procedures and functions to use in your own programs. As an example, we will construct a program for you now, step by step. But, before we start, you should store all the procedures and functions on a disk; your own library disk (they are all on *Today Disk #15*).

This will be a simple program to read all the records from the *Doctor Who* data file and print out the show names (see the *Doctor Who* article on page 26. You must have the data base file on disk to use this program). To make the program more friendly, it starts with an information screen. Then it displays the shows. First issue the **NEW** command, then **AUTO** to provide the line numbers for you:

```
new
auto
0010 info
0020 display'shows
0030 //
0040 proc info
0050 endproc info
0060 //
0070 proc display'shows closed
0080 endproc display'shows
```

So far the program won't do anything, but an important start has been made. We have an outline of what the program will do. The first two lines provide the program. After that come the procedures, which are "empty" for now. They will be filled in as we determine what is necessary to implement them.

Before we go further, let me remind you of a special convention that has been used to determine the last record in the random file *ran.doctorwho*. The first record contains the number of the last show. The first show is actually stored in record number 2. Thus, the show number will always be one less than the number of the record it is stored in. Once this is understood, we can proceed to fill in the **display'shows** procedure:

```
71 open file 2,"ran.doctorwho",random 248
72 for show:=1 to last'show do
73   record=show+1//adjust
74   read file 2,record:show'name$
75   print show'name$
76 endfor show
77 close file 2
renum
```

[Note: I am inserting lines in a logical order based on their need to accomplish the programming task. Those who find this confusing may type in the finished program at the end of the article, and just observe how the program progressed into its final form.]

Since our original "empty" **display'shows** procedure started at line 70 and ended at line 80, lines 71-77 are inserted into the procedure. They are a start for what needs to be done to display the show names. Note that **random 248** is the record length specific to the data file *ran.doctorwho* (clearly explained in the **Doctor Who** article on page 26). Some key items are still missing from the procedure: **last'show** is not defined and **show'name$** is not dimensioned.

Since we issued a **RENUM** command after adding lines 71-77 to the **display'shows** procedure, our program now looks like this:

```
0010 info
0020 display'shows
0030 //
```

More ►

```
0040 proc info
0050 endproc info
0060 //
0070 proc display'shows closed
0080  open file 2,"ran.doctorwho",random 248
0090  for show:=1 to last'show do
0100   record=show+1//adjust
0110   read file 2,record:show'name$
0120   print show'name$
0130  endfor show
0140  close file 2
0150 endproc display'shows
```

Now let's add a line at the start of the procedure to dimension the variable **show'name$** to 27 characters. This is the maximum length already determined by the **Doctor Who Data Base Program**. Next add one line right after we open the file (line 80) to read the number of the last show stored in record #1. Type in these lines:

```
75 dim show'name$ of 27
85 read file 2,1:last'show
```

You can now run the program and it will display all of the show names. However, a couple of things can be done to make the listing easier to read. The show numbers could be included and a pause feature inserted to let the reader finish reading the names before they scroll off the screen. The print statement (line 120 as shown above), can be changed to include the show number. The **PRINT USING** statement below, displays show numbers as three digits plus a space. And right after we print a show name, let's include a pause:

```
120 print using"### "+show'name$: show
125 shift'wait
```

Shift'wait is a good procedure that allows a user to pause a program as necessary. It pauses until the shift key is pressed. However, if the <*shift lock*> is down, no detectable delay will occur. Since the **shift'wait** procedure is included on our library disk, we can merge it in now:

```
enter "shift'wait.proc"
```

This is a good time to save our program. It isn't finished yet, but we have done too much work to risk losing what we have already done.

```
save "temp1"
```

We have finished the procedure to display the shows. Now we can fill in the **info** procedure. The screen should usually be cleared before anything gets printed at the start of a program. **Page** does this quite well. We don't need to say a lot about the program, but it would be nice if the text wasn't crammed at the top of the screen. **Print'at** places text anywhere on the screen. Both **page** and **print'at** are included on the library disk.

Looking at our program so far, we see that the **info** procedure starts at line 40. Since it is "empty" so far, it ends at line 50. Add these lines to it:

```
41 page
42 print'at(5,1,"Dr. Who Shows Display")
43 print'at(8,1,"Using CT#15 Data Base")
44 print // provide carriage return
```

Now, both **page** and **print'at** procedures must be merged into the program:

```
renum
enter "page.proc"
renum
enter "print'at.proc"
renum
```

It is important that each procedure you take from the library disk be renumbered

More ►

before you merge in another one. This is because every procedure in that library starts at line 9000. **RENUM** prevents lines overwriting each other.

Our program is now almost finished. It works the way we want it to, but still has one potential problem. This program might be run from a disk which doesn't contain the data base. To minimize this problem, we must check to see if the data base is on the disk before we call the procedure to display it. This can be accomplished with an **IF** statement and the **file'exists** function in place of line 20 which just assumes the file is there and calls for the display. Type these lines:

```
20 if file'exists("ran.doctorwho") then
21  display'shows
22 else
23  print'at(12,1,"file not found")
24 endif
```

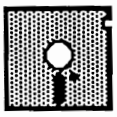
Now, merge in the **file'exists** function from the library disk, renumber the program neatly, and save the program:

```
renum
enter "file'exists.func"
renum
save "print'dr'who"
```

The program is now finished and ready to run. Notice that using this method of writing our program, over half the program was already written for us and only needed to be merged from disk. Another example of program construction is at the start of Appendix D in the *COMAL Handbook*. Even better, the book *Captain COMAL Gets Organized* is designed to stress modular programming, and comes with its own disk containing the "modules" required to construct its programs. The final version of the program is also on *Today Disk #15*:

```
    info
==> if file'exists("ran.doctorwho") then
!       display'shows
+-> else
!       print'at(12,1,"file not found")
--> endif
    //
==> proc info
!       page
!       print'at(5,1,"dr. who shows display")
!       print'at(8,1,"using ct#15 data base")
!       print // provides carriage return
--> endproc info
    //
==> proc display'shows closed
!       dim show'name$ of 27
!       open file 2,"ran.doctorwho",random 248
!       read file 2,1: last'show
!   ==> for show:=1 to last'show do
!   !       record:=show+1 // adjust
!   !       read file 2,record: show'name$
!   !       print using "### "+show'name$: show
!   !       shift'wait
!   --> endfor show
!       close file 2
--> endproc display'shows
    //
==> proc shift'wait closed
!       shift'flag:=653
!   ==> while not peek(shift'flag) and not esc do
!   !       print "press shift"+chr$(145)
!   !       print "           "+chr$(145)
!   --> endwhile
--> endproc shift'wait
    //
==> proc page closed
!   ==> if peek(152)=0 then
!   !       print chr$(147), //clear screen
!   +-> else // for printer
!   !       print chr$(12), //form feed
!   --> endif
--> endproc page
    //
==> proc print'at(row,col,text$) closed
!       addr:=1024+(row-1)*40
!       poke 209,addr mod 256
!       poke 210,addr div 256
!       poke 211,col-1
!       poke 214,row-1
!       print text$,
--> endproc print'at
    //
==> func file'exists(filename$) closed
!       dim ds$ of 2
!       open file 79,filename$,read
!       ds$:=status$
!       close file 79
!       return ds$="00"
--> endfunc file'exists
```

Edit Random File



by Richard Bain

Edit'wheel'data, on *Today Disk #15*, was written as a companion program to the *wheel'of'fortune* game. With a few small changes, it can be used to edit other random files with one string per record.

When **RUN**, it first asks for a filename. The default prompt, *ran.wheel*, is the file used with the *wheel'of'fortune* game (see page 56). The file is then opened:

open file 2,filename$+",r",random 41

The desired file may not be on the disk because the wrong disk is in the drive or the filename was typed incorrectly. Normally, when COMAL tries to **OPEN** a **RANDOM** file that is not on the disk, it will create a new file entry. This is often undesirable. It is important to check if the file exists before trying to open the file, or include +",r". Then the disk drive will respond with a *file not found* error rather than create a new file entry. This is a faster method to open an existing file because the file is only opened once, not twice as with the *file'exists* method. A second advantage to this method is that the disk drive will give a *file type mismatch* error if the file is on the disk, but is not a **RANDOM** file. A disadvantage is that it is not transportable to other versions of COMAL.

Random 41 specifies that the file record length is 41 bytes. This allows for 39 characters plus 2 bytes for the character count as used by *ran.wheel*. Other files may need a different record length. The maximum possible length is 254.

Edit'wheel'data then reads in the entire file. The file contains a special format to make this easy. The first record in the file contains, **max'used**, the number of the last valid record. Once **max'used** is determined, records 2 through **max'used** can safely be **READ**. The program must keep track of the number of records and **WRITE** it to the first record prior to closing the file. This can serve as a useful standard for all **RANDOM** files.

The program then asks if you want to edit the data. This option lets you change any existing record. The editing process is easy. The old entry is printed on the screen and the cursor is placed on the first character. Press *<return>* to keep it unchanged, or type in the changes you desire. Note that strings are limited to 39 characters.

After you are finished editing the data, you have the option of adding more data (up to 100 records). Type in the names and phrases you want. When you have entered all the data, hit *<return>* on a blank line to tell the computer you are finished.

Next, you will be asked if you want any more changes. I recommend you say yes. This will let you edit the data again. Even if you think it is right, this option lets know before it is saved.

When the data is the way you like it, you will be asked if you want to save it. All changes have been made in computer memory only, so you almost certainly will want to save it back to the disk. The default filename is the one specified at the start of the program, but can be changed. If you specify a filename that is not on the disk, a new file will be created. *[Note: it is possible to swap disks during the program, so this option can be used to copy the file from one disk to another. This may be important for those of you who don't have a RANDOM file copy program.]*

More ►

The records are written to the file in reverse order. When creating a new **RANDOM** file, or appending to an old one, it is important to **WRITE** the last record before the others. Then the file only needs to allocate new records once instead of allocating each record as it is used.

Random File Problems

It seems that if you **PRINT FILE** a number to a **RANDOM** file in COMAL 0.14, the record entry will start with **CHR$(13)** and then be padded with **CHR$(0)**s. In COMAL 2.0, the number is placed in the record, but the rest of the record is still padded with **CHR$(0)**s, overwriting anything which may have been there before. The problems seem to disappear when using **READ FILE** and **WRITE FILE**.

The 1541 has known bugs writing to **RANDOM** files. I try to avoid them by reading back each record right after it is written. This seems to solve the problem. If it doesn't, at least the program stops when the error occurs.

```
    dim msg$ of 40, filename$ of 22
    dim phrase$(100) of 39, keyin$ of 1
    page
    print tab(25);"ran.wheel"+chr$(142)+chr$(8)
    input chr$(145)+"file to be edited: ": filename$
    open file 2,filename$+",r",random 41
    check'disk
    read file 2,1: max'used
==> for x:=2 to max'used do
!       read file 2,x: phrase$(x)
!       print phrase$(x)
--> endfor x
    close file 2
==> repeat
!       edit'data
!       if max'used<100 then add'data
!       print
!       input "any changes? (y/n) ": keyin$
--> until keyin$ in "nN"
    write'data
    //
==> proc edit'data
!       input "want to edit data? (y/n) ": keyin$
!   ==> if keyin$="y" or keyin$="Y" then
!   !       page
!   !       print "change data - hit return to keep it"
!   !   ==> for x:=2 to max'used do
!   !   !       print phrase$(x)
!   !   !       input chr$(145): phrase$(x)
!   !   --> endfor x
!   --> endif
--> endproc edit'data
    //
==> proc add'data
!       input "do you want to add data? (y/n) ": keyin$
!   ==> if keyin$="y" or keyin$="Y" then
!   !       page
!   !       print "type in data - hit return to quit"
!   !   ==> repeat
!   !   !       max'used:+1
!   !   !       input "": phrase$(max'used)
!   !   --> until phrase$(max'used)=" " or max'used=100
!   !       if phrase$(max'used)=" " then max'used:-1
!   --> endif
--> endproc add'data
    //
==> proc write'data
!       input "want to save this? (y/n) ": keyin$
!   ==> if keyin$="y" or keyin$="Y" then
!   !       page
!   !       print tab(17);filename$
!   !       input chr$(145)+"filename: ": filename$
!   !       pass "i0"
!   !       open file 2,filename$,random 41
!   !       check'disk
!   !   ==> for x:=max'used to 2 step -1 do
!   !   !       write file 2,x: phrase$(x)
!   !   !       read file 2,x: phrase$(1)
!   !   !       print phrase$(x)
!   !   !   ==> if phrase$(1)<>phrase$(x) then
!   !   !   !       print "rel file error."
!   !   !   !       close file 2
!   !   !   !       stop
!   !   !   --> endif
!   !   --> endfor x
!   !       write file 2,1: max'used
!   !       read file 2,1: x
!   !   ==> if x<>max'used then
!   !   !       print "rel file error."
!   !   !       close file 2
!   !   !       stop
!   !   --> endif
!   !       close file 2
!   --> endif
--> endproc write'data
    //
==> proc page
!       print chr$(147),
--> endproc page
    //
==> proc check'disk
!       msg$:=status$
!   ==> if msg$(1:2)<>"00" then
!   !       print msg$
!   !       close file 2
!   !       stop
!   --> endif
--> endproc check'disk
```

Wheel Of Fortune Revisited

by Richard Bain 

The Wheel of Fortune is the most popular game show on television. It is becomming one of the most popular game shows in COMAL too. In *COMAL Today #13* we released Bob Hoerter's 2.0 version of the game. I felt our COMAL 0.14 users also deserved a version. This was easy to do because I had already written one independently of the other. I tried to make my game as close to the television game as possible. It is on *Today Disk #15*.

Wheel'of'fortune can be played by one person or many. There are four rounds; the winner goes to the bonus round. One special feature of the game is the option of having a host. Normally, no host is needed since an external file can supply phrases for you to guess. This can be the file supplied on *Today Disk #15* or one that you create using the random file editor on page 54. However, once you play the game awhile, you may find that you get the same puzzles over again. One alternative is to let one of the players be a host. The host types in the puzzles, so the other players continually get new ones. Of course the host doesn't get to guess the puzzles. One or two players can use puzzles supplied by the computer. Large groups can have different people take turns being host.

I added extra features to the game to make it more fun. When you spin the wheel, you actually see a wheel. I couldn't make the wheel spin, due to limited speed and memory of COMAL 0.14, so I made it like a roulette wheel. A ball rolls around the wheel until it gradually comes to a stop. (Hope it makes it past bankrupt and lose your turn.) Sound effects make this even more realistic.

RULES

When the game starts, decide if there is a host. Then say how many players there are and their names (the host is not considered to be a player). There are four rounds. The players rotate starting a round. During your turn, you have three options: spinning the wheel, buying a vowel, or solving the puzzle (follow the prompts to see how to do this).

After spinning the wheel, you will see *bankrupt*, *lose your turn*, or an amount of money between $200 and $1000. *Bankrupt* means your lose all your money from the current round, but you keep winnings from the previous rounds. You also lose your turn. *Lose your turn* means just that, but you keep your money. If you see a dollar amount, then type a letter (vowels are not accepted). If the letter is in the puzzle and has not been guessed before, you get money for each time the letter appears. If the letter has been guessed before or is not in the puzzle, you lose your turn. Keep an eye on the used letter board.

It costs $250 to choose a vowel. If you choose a vowel but don't have enough money, or if the vowel you guess is not in the puzzle, you lose your turn. If the vowel you guess is in the puzzle, your turn continues.

To solve the puzzle, you must type in the answer exactly as it appears in the solution. If you misspell the answer or make a typing mistake, you lose your turn. Double check that you typed what you intended before hitting *<return>*. If you solve the puzzle correctly, you get to keep all the money you won during the round; there is a $200 house minimun. Sorry, but there is no shopping option.

More ►

After the four rounds, the player with the most money goes to the bonus round (a winner is chosen randomly in the case of a tie). In the bonus round, you get to choose 5 consonants and a vowel. After they are exposed you must solve the puzzle for $10,000. There is no time limit.

Using the Program

Wheel'of'fortune will work perfectly if **RUN** from *Today Disk #15*. If you plan to copy it to another COMAL disk, be sure to include these files: *wheel'of'fortune*, *hrg.wheel*, *ran.wheel*, and *load/save.mem*.

WARNINGS

The computer is intolerant of mistakes. If you type the wrong letter, the computer won't bite you, but it won't let you fix your mistake either. (I've never seen a contestant on the game show get a second chance). The moral is: be careful how you type. Also note that different sections of the program expect different replies. If the computer prompts you for a consonant and you type a vowel, nothing will happen. This is normal, don't take your computer back to the store thinking something is wrong.

One final note: the function keys are used to choose certain options. This shouldn't cause any trouble, but if pressed at the wrong time, the function keys may swap the text and graphics screen. If you see the wheel and the ball isn't moving, try hitting F1 to go back to the text screen.

[Note: the version of wheel'of'fortune listed here is a stripped down version of the game on the disk. It lacks the spinning wheel feature which involves an external file which can not easily be listed in the newsletter.]

```
instructions
initialize
for round:=1 to 4 do
 get'phrase(host,phrase$)
 pencolor 6
 print'at(7,17,"round "+chr$(round+48))
 pencolor 2
 print'at(23,12,"used letter board")
 player:=(round-1) mod players+1; solved:=false
 repeat
  pencolor 6
  print'at(11,9,names$(player)+" $")
  take'turn(player,players,money,phrase$,used'letters$,solved,
  angle) // wrap line
  player:=player mod players+1
 until solved
endfor round
page
pencolor 6
print'at(5,13,"wheel of fortune")
print chr$(13)
for l:=1 to players do print tab(10),names$(l);"won $",mon
ey(2,l) // wrap line
find'winner(player,names$,money,players)
bonus'round(names$(player),money(2,player),host,phrase$)
print chr$(9)
///
proc instructions closed
 page
 pencolor 6
 border 3
 background 1
 print chr$(14) // lower case
 print'at(2,13,chr$(14)+"Wheel of Fortune")
 print'at(4,13,"by Richard Bain")
 print'at(6,5,"This game is played like the TV")
 print
 print "game show. Follow the prompts and enjoy."
 print "Note: The host sets the names or phrases",
 print "          for the contestants."
 print "       The host is not a contestant."
 print "       If there is not a host, a phrase"
 print "          will be supplied automatically.",
 print "       There are 4 rounds."+chr$(13)
 print "       Vowels cost $250."
 print "       Consonants are worth from $200 to"
 print "          $1000 each."
 print "       Beware of Bankrupt and"
 print "          Lose Your Turn."
 print "       There is a $200 house minimum for"
 print "          solving a puzzle."
 print'at(25,8,"press any key to continue")
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc instructions
///
proc initialize
 dim letter$ of 1, phrase$(3) of 39, used'letters$ of 26
 dim guess$ of 39, space$ of 39
 page
 solved:=false
 space$(1:39):="" // 39 spaces
```

More ►

```
used'letters$:=space$(1:26)
host:=rnd(-65536*peek(160)-256*peek(161)-peek(162))
page
print chr$(142) // upper case mode
print chr$(8) // keep it
print'at(5,13,"wheel of fortune")
print'at(7,12,"is there a host? ")
input "": guess$
host:=guess$="y"
print'at(9,9,"how many contestants? ")
input "": players
dim money(2,players), names$(players) of 13
print
for l:=1 to players do
 money(1,l):=0; money(2,l):=0
 input "player "+chr$(l+48)+", what is your name? ":
 names$(l)(1:13) // wrap line
endfor l
endproc initialize
///
proc page
 print chr$(147)
endproc page
///
proc take'turn(p,n'p,ref m(,),ref phrase$(),ref used$,ref s,ref
 ang) closed // wrap line
 dim guess$ of 39, space$ of 39
 space$(1:39):="" // 39 spaces
 repeat
  init
  get'key(guess$,chr$(133)+chr$(134)+chr$(135))
  print'at(14,1,space$+chr$(13)+space$+chr$(13)+space$)
  case ord(guess$) of
  when 133
   spin // f1
  when 134
   buy // f3
  otherwise
   solve // f5
  endcase
  if not turn'over then pause(2000)
 until turn'over
 if s then
  for count:=1 to 3 do bell(75,32,100,10)
 else
  bell(8,32,300,0)
 endif
 pause(2000)
endproc take'turn
///
proc init
 pencolor 5
 print'at(14,9,"press f1 to spin")
 print'at(15,9,"press f3 to buy a vowel")
 print'at(16,9,"press f5 to solve puzzle")
 print'at(19,1,space$)
 print'at(21,1,space$)
 turn'over:=false
 pencolor 6
 print'at'num(11,24,m(1,p))
endproc init
///
proc pause(delay) closed
 for count:=1 to delay do null
endproc pause
///
proc print'at(r,c,s$) closed
 poke 209,(1024+40*(r-1)) mod 256
 poke 210,(1024+40*(r-1)) div 256
 poke 214,r-1
 poke 211,c-1
 print s$,
endproc print'at
///
proc print'at'num(r,c,n) closed
 poke 209,(1024+40*(r-1)) mod 256
 poke 210,(1024+40*(r-1)) div 256
 poke 214,r-1
 poke 211,c-1
 print n;"  ",
endproc print'at'num
///
proc bell(tone,wave,duration,skip) closed
 sid:=54272
 poke sid+1,tone
 poke sid+3,8
 poke sid+5,136
 poke sid+6,136
 poke sid+24,15
 poke sid+4,wave+1
 pause(duration)
 poke sid+4,wave
 poke sid+24,0
 pause(skip)
endproc bell
///
proc get'key(ref guess$,string$) closed
 while key$<>chr$(0) do null
 repeat
  guess$:=key$
 until guess$ in string$
endproc get'key
///
proc spin
 pencolor 0
 dollars:=cash
 case dollars of
 when 100
  print'at(19,1,"bankrupt")
  m(1,p):=0
  pencolor 6
  print'at'num(11,24,0)
  turn'over:=true
 when 150
  print'at(19,1,"lose your turn")
  turn'over:=true
 otherwise
  print'at(19,1,"guess a letter for $")
  print'at'num(19,21,dollars)
  get'key(guess$,"bcdfghjklmnpqrstvwxyz")
  new'letter
 endcase
endproc spin
///
proc buy
 pencolor 0
```

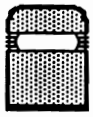
More ►

```
 if m(1,p)<250 then
  print'at(19,1,"you can't afford a vowel")
  turn'over:=true
 else
  m(1,p):-250
  print'at(19,1,"guess a vowel")
  pencolor 6
  print'at'num(11,24,m(1,p))
  get'key(guess$,"aeiou")
  dollars:=0
  new'letter
 endif
endproc buy
///
proc solve
 pencolor 0
 print'at(19,1,"please type your guess:")
 print'at(21,1,"")
 input "": guess$
 if guess$=phrase$(1) then
  pencolor 2
  print'at(9,((40-len(phrase$(2))) div 2+1),phrase$(1))
  if m(1,p)<200 then m(1,p):=200
  m(2,p):+m(1,p)
  pencolor 4
  print'at(23,12," you won $")
  print'at'num(23,22,m(1,p))
  for n:=1 to n'p do m(1,n):=0
  used$:=space$(1:39)
  s:=true
 endif
 turn'over:=true
endproc solve
///
proc new'letter
 if guess$ in used$ or not guess$ in phrase$(1) then
  turn'over:=true
 else
  for l:=1 to len(phrase$(2)) do
   if guess$=phrase$(1)(l) then
    phrase$(2)(l):=guess$
    m(1,p):+dollars
    bell(64,64,200,50)
   endif
  endfor l
  pencolor 4
  print'at(9,((40-len(phrase$(2))) div 2+1),phrase$(2))
  pencolor 6
  print'at'num(11,24,m(1,p))
 endif
 used$(ord(guess$)-64):=guess$
 pencolor 2
 print'at(25,8,used$)
endproc new'letter
///
proc get'phrase(host,ref phrase$()) closed
 page
 pencolor 6
 print'at(5,13,"wheel of fortune")
 for count:=1 to 3 do phrase$(count):=""
 if host then
  pencolor 5
  print'at(7,1,"type a phrase:")
  print'at(9,1,"")
  input "": phrase$(1)
 else
  read'phrase(phrase$(1))
 endif
 for l:=1 to len(phrase$(1)) do
  if not phrase$(1)(l) in "abcdefghijklmnopqrstuvwxyz" then
   phrase$(2)(l):=phrase$(1)(l)
   phrase$(3)(l):=phrase$(1)(l)
  else
   phrase$(3)(l):="Z" // diamond
   phrase$(2)(l):=chr$(29) // crsr  right
  endif
 endfor l
 page
 pencolor 12
 print'at(9,((40-len(phrase$(1))) div 2+1),phrase$(3))
 pencolor 4
 print'at(9,((40-len(phrase$(1))) div 2+1),phrase$(2))
endproc get'phrase
///
proc read'phrase(ref phrase$) closed
 dim file'name$ of 40, msg$ of 40
 repeat
  page
  pencolor 5
  print'at(10,1,"what is the filename? rnd.wheel")
  print'at(10,23,"")
  input "": file'name$
  open file 2,file'name$+",r",random 41
  msg$:=status$
  if msg$(1:2)="00" then
   read file 2,1: count
   count:=rnd(2,count)
   read file 2,count: phrase$
   msg$:=status$
  endif
  close file 2
  if msg$(1:2)<>"00" then
   pencolor 2
   print chr$(13)+msg$+chr$(13)
   pencolor 0
   print "did you type the filename correctlly?"
   print "is the correct game disk in the drive?"+chr$(13)
   pencolor 6
   print "      press any key to try again"
   while key$<>chr$(0) do null
   while key$=chr$(0) do null
  endif
 until msg$(1:2)="00"
 page
endproc read'phrase
///
func cash closed
 dim conversion$ of 20
 angle:=rnd(1,20)
 conversion$:="knfsdgoaemrlhaqjcpib"
 return (ord(conversion$(angle))-63)*50
endfunc cash
///
proc find'winner(ref winner,ref names$(),ref money(,),players
 ) closed // wrap line
 winner:=1; high'score:=money(2,1)
```

More ►

```
 for player:=2 to players do
  if money(2,player)>high'score then
   high'score:=money(2,player)
   winner:=player
  elif money(2,player)=high'score then
   if rnd(0,1) then winner:=player
  endif
 endfor player
 print
 pencolor 4
 print "congratulations";names$(winner)
 pencolor 2
 print'at(25,2,"press any key to play the bonus round.")
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc find'winner
///
proc bonus'round(name$,money,host,ref phrase$()) closed
 dim letters$ of 6, letter$ of 1
 get'phrase(host,phrase$)
 pencolor 0
 print'at(11,1,"guess 5 consonats, "+name$)
 print'at(7,1,"")
 pencolor 5
 for l:=1 to 5 do
  get'key(letter$,"bcdfghjklmnpqrstvwxyz")
  print letter$,
  letters$(l):=letter$
 endfor l
 pencolor 0
 print'at(13,1,"guess a vowel, "+name$)
 get'key(letter$,"aeiou")
 letters$(6):=letter$
 pencolor 5
 print'at(7,7,letter$)
 for l:=1 to 6 do
  for l2:=1 to len(phrase$(2)) do
   if letters$(l)=phrase$(1,l2) then
    phrase$(2,l2):=letters$(l)
    bell(64,64,200,50)
   endif
  endfor l2
 endfor l
 pencolor 4
 print'at(9,((40-len(phrase$(2))) div 2+1),phrase$(2))
 pencolor 6
 print'at(15,1,"for $10,000, solve this puzzle.")
 print'at(17,1,"good luck "+name$)
 pencolor 2
 print'at(19,1,"")
 input "": phrase$(3)
 print'at(9,((40-len(phrase$(1))) div 2+1),phrase$(1))
 if phrase$(1)=phrase$(3) then
  for count:=1 to 10 do bell(75,32,100,10)
  pencolor 4
  print'at(21,1,"way to go, "+name$+chr$(13))
  print "you won $",10000+money
 else
  bell(8,32,300,0)
  print'at(21,1,"sorry, "+name$+chr$(13))
  print "you still won $",money
 endif
endproc bonus'round
```

Skyview



by Lowell Zabel

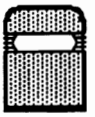
This program calculates and plots on the screen the position of the sun, planets and constellations. Any location, time and date (after 1960) may be selected. For plotting, the sky is separated into four horizons: East, West, South and North. Altitudes from 0 to 80 degrees above the horizon are recognized. The plots use rectangular coordinates to represent a hemisphere, therefore the higher altitudes are expanded and the constellations are somewhat distorted, but still recognizable. The sun is represented by the asterisk (*), planets by their initials and stars by dots (Mercury is a lower case **m**, Mars is an upper case M).

All horizons are plotted white on black (adding a few lines could make the daylight sky blue and night sky black). I have tried to make the on-screen instructions sufficient so that no further help is required. A time and location when all of the planets and the sun are simultaneously above the horizon are: Year: **1981**, Month: **11**, Day: **26**, Time: **11.5**, Latitude: **38.24**, Longitude: **122.49**, Horizon: **South**, Time Zone: **8**. *Eastern Standard Time is zone 5, Pacific time is zone 8.*

The constellations included are: Adromeda, Aquarius, Aquila, Aries, Auriga, Bootes, Cancer, Canis Major, Canus Minor, Capricornus, Cassiopeia, Cepheus, Cepheus, Cetus, Corona, Corvus, Crater, Cygnus, Delpninus, Draco, Eridanus, Gemini, Hercules, Hydra, Leo, Libra, Lyra, Ophiuchus, Orion, Pegasus, Perseus, Pisces, Sagittarius, Scorpio, Serpens, South Polar Region, Taurus, Triangulum, Ursa Minor, Ursa Major, Virgo

The program is on *Today Disk #15.* ■

Instant Help Screens



by Dick Klingens, Dutch COMAL Users Group

These days, user friendly programs are a must. COMAL provides us a way to connect a lot of help screens to your programs. This first example uses **GETSCREEN** and **SETSCREEN** from the **SYSTEM** package.

```
INPUT "Help? (1-5)": n
help(n)
//
PROC help(num) CLOSED
 USE system
 DIM this'screen$ OF 1505, help'screen$ OF 1505
 OPEN FILE 9,"help"+STR$(num),READ
 READ FILE 9: help'screen$
 getscreen(this'screen$)
 setscreen(help'screen$)
 CLOSE FILE 9
 INPUT AT 24,1: "Type [RET] ":c$
 setscreen(this'screen$)
ENDPROC help
```

But the help screens are **slowly** loaded from disk, and take a lot of memory (7525 bytes) if all the help files are read into an array, dimmed with:

DIM helpscreen$(5) OF 1505

Avoid loading time and loss of memory by saving the help screens **with** your program. It can be done with the *pkg.text* on *Today Disk #15*. Suppose you have 5 help screens on disk in the files *help1* to *help5*. Now link *pkg.text* onto your program:

LINK "pkg.text"

Add the following program lines:

```
PROC add'helps(n) CLOSED
 USE text
 DIM screen$ OF 1505
 rewrite
 FOR t:=1 TO n DO
  OPEN FILE 9,"help"+STR$(t),READ
  READ FILE 9: screen$
  writeln(screen$)
  CLOSE FILE 9
 ENDFOR t
ENDPROC add'helps
```

Now type:

SCAN
add'helps(5)
SAVE "file.help"

The screens were saved with your program. Now reading the screens is easy:

```
INPUT "Help? (1-5)": n
help(n)
//
PROC help(num) CLOSED
 USE text
 USE system
 DIM this'screen$ OF 1505
 DIM help'screen$ OF 1505
 getscreen(this'screen$)
 read'screen(num,help'screen$)
 setscreen(help'screen$)
 INPUT AT 24,1: "Type [RET] ":c$
 setscreen(this'screen$)
 //
 PROC read'screen(screen'num,REF screen$) CLOSED
  // sequential read from text buffer
  reset
  FOR skip:=1 TO screen'num DO readln(screen$)
 ENDPROC read'screen
ENDPROC help
```

Because the number of bytes free in the package is more than 15050, a total of 10 screens can be saved with one program. See the program *demo/text*, also on *Today Disk #15*.

The text package, version 1.02, has been updated to include bytes'free and the version functions.

Further reference:

Text Package, COMAL Today #11, page 63 ■

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

DRAW UNIVERSE

Draw'universe on *Today Disk #15* is an interesting program which draws a beautiful night sky. It shows spiral galaxies and moving comets in a colorful dynamic fashion. It was submitted by James Adams. ■

COMAL Standards Meeting

by Brian Grainger
Introduction by TeleNova

[Editors note: everyone knows that we need a COMAL Standards Group - but since little is heard about them, stories like this can result. Reprinted from ICPUG, Vol 8 No 5.]

The story so far ...

In the (BASIC) beginning was the word, and the word was GOTO...

Now, Borge (Christensen) looked upon the word and saw that it was bad (or at least if not all bad, then certainly much overused). And Borge did address the multitude saying, "Behold! I have invented a new word, and the word is PROC!"

Now the people looked upon this word, and they saw that it was good. Thus they embraced the word, and built a new language to house it.

And it came to pass that a True COMAL Deity entered upon the scene and established a group of disciples who were all similar worshippers of the one True Language.

Now for many years this group did use up much company expenses in mutual back-slapping and (liquid) celebration of The Language.

But lo! They did not notice that the BASIC reason for The Language was disappearing because it had built up even more followers, who were not all living on expense accounts. And so it came to pass that the True COMAL Deity lost interest in the language, saying "whosoever wishes to lead the worship may do so, but there's nothing in it for me". And there was much wailing and gnashing of teeth, and the disciples spoke amongst themselves saying "why has our God forsaken us, and what's more, how are we going to preserve our biannual expense account jollies?"

And so the disciples travelled to the land of Bagpipes and Sporrans (and more importantly, Scotch) in order to debate these matters, hoping that it wouldn't be the last time, but making sure that if it was they made the most of it...

That is a humorous account of events up to the time of the 1985 COMAL Standardization meeting as written by representatives of TeleNova, a Swedish company. The rumours of Borge deserting COMALites were exaggerated somewhat, as were the thoughts that the 1985 meeting may have been the last. In August, the 1986 Standardization meeting was held in Denmark and I accepted the invitation to attend, representing users of COMAL in England. I hasten to add it was at my own expense.

I went to Denmark for the two and a half days of meetings, not sure of what to expect but with eager anticipation. At long last I was participating in how the language would develop.

There are two groups involved, the Standardization Group, and the Development Group. The latter consists of all those attending and the purpose is to discuss the various proposals that may be put forward for changes to COMAL. In order for the Development Group to accept a proposal, two thirds of members present must support it. Agreement on a proposal means that it can be carried forward to the Standardization Group, where the ultimate decision is made. The Standardization Group consists of implementors of COMAL only. Other attenders, such as myself, can only

More ►

observe at the Standardization Group meetings. We have no voting rights. For the Standardization Group to accept a proposal, four fifths of members present have to support it.

The majorities required to get proposals accepted may seem rather high but this way ensures the COMAL language does not get changed on a whim. Certainly there is no universal agreement among the implementors on how the language should develop. It is my feeling that because an almost universal level of support is required to accept a proposal, the user is protected. User programs do not become obsolete as the language develops due to lack of compatibility between one version of the language and the next.

In Denmark, during the three days, there was an attendance of around 20, from 5 countries, representing 6 implementors, and the users of England, Scotland, and Eire. In addition, the writers of the manual for the COMAL cartridge (*Cartridge Tutorial Binder*) put in an appearance, as did Borge himself.

Having only come into contact with one implementor previously, UniComal, who produced the COMAL cartridge for the C64, it was fascinating to meet some of the others. The views expressed varied widely. The very conservative wanted to make few changes to the language in case they made ANY existing program obsolete. On the other hand, the radicals wanted to make lots of changes so as to provide facilities in COMAL for any purpose imaginable! It is amazing that anything gets agreed at all, but the result of the meeting was to steer a course between the two extremes, with some additons being agreed and more to be put to a vote at a later time.

A word about the various implementors of COMAL who were present may be of interest to those who think that COMAL is limited to Commodore machines.

UniComal need no introduction. Of all the implementors, they have decided to go for the international market with versions for Commodore and IBM.

RegenCentralen (RC) who have implemented COMAL on the RC Piccolo, a machine much used in Danish Schools.

TeleNova who have implemented COMAL on Compis computers which are mainly used in Sweden.

Dansk Data Elektronik have implemented COMAL under the Unix operation system, but, again, seem to be limited to Denmark.

Dench Systems are a company working with UniComal developing COMAL in real-time applications.

Finally, and the biggest surprise to me, was Dalgasoft, who have COMAL implemented on the Amstrad 464 and 6128 and are working on a CP/M version.

John Clack, a user representative at the meetings, uses COMAL on a VAX.

Apart from the VAX version, all the versions were on display in what became known as the Hornstrup Computer Centre (Hornstrup being the place where the meeting took place). One does tend to get rather isolated in one's thinking in using only Commodore machines, and an important outcome of the meeting for me was to see other versions being used. The subtle differences could be weighed up, but the value of a standard and usefulness of the

More ►

group becomes clear when you can type exactly the same program into a Commodore 64, IBM PC, Amstrad 464, Piccolo, or Compis and they all work!

The meetings discussed 22 proposals, mostly for additions to the COMAL extensions. They ranged from the simple, such as introducing **a$:+b$** as a shorthand for **a$:=a$+b$** to the complex, such as the ability to address subsets of an array with one variable descriptor. One can guess how complex the discussions became by the results. The Development Group passed 8 of the proposals, having amended the syntax on 2 of the 8. It threw out 13 and liked the idea of the remaining one, but asked for the syntax to be reconsidered. The Standards Group then passed 2 of the 8 with the others going to a postal vote since the group had not had sufficient warning of the proposals.

So, what were the two agreed proposals that appear to be the only achievements of two and a half days?

(1) The ability to assign the same value to every element of an array with a single statement. Example:

```
a(,):=1
```

(2) The ability to PRINT or INPUT whole arrays by one statement. Example:

```
DIM a(10,10)
a(,):=1
PRINT a(,);
```

This would print the 100 elements of the two dimensional array (set to the value of 1) with a single space between each value.

In reality, the meeting achieved much more than this. Apart from the six items awaiting the outcome of a postal vote, important discussions took place on the future of COMAL regarding (a) packages; (b) data structures especially record structures and pointers; and (c) real time and concurrency. In the fullness of time, after careful consideration, these items will form the basis of future enhancements that will widen the usage of COMAL. In view of my enthusiasm for graphics, I was also roped into submitting a proposal for a defined graphics extension (*also see the preliminary Graphics Kernal report in COMAL Today #7*, page 8).

After all the discussions the only business to conclude was when and where to hold the next meeting. It is likely to be in the Spring of 1987 and, one hopes, in London.

I had a great three days (plus one rest day). At one point, I remember the incredible amount of time it took, and the amount of heat generated in discussing:

"What value should be returned by the expression A$ IN B$ when A$ is the empty string?"

Possible answers include:

(a) LEN(B$)+1
(b) 0
(c) any positive number
(d) undefined

The reasoning behind (a) is that the empty string is regarded as part of every string and therefore A$ IN B$ must be TRUE (a value not equal to 0) when A$ is empty.

The reasoning behind (b) is that it is hard to explain why the following will not work on an IBM PC if (a) is used:

More ►

```
DIM a$ OF 2
PRINT "Shall I reformat the harddisk?"
REPEAT
  a$=KEY$
UNTIL a$ IN "yYnN"
IF a$ IN "yY" THEN PASS "FORMAT C:"
```

(On an IBM PC, KEY$ returns an empty string when no key is pressed! On a C64, CHR$(0) is returned!)

Further Reference:

COMAL Implementations, COMAL Today #11, page 10
COMAL Standards Conference, COMAL Today #7, page 8 ■

PSALM 23

COMAL is my language; I shall not want.

It maketh me to structure my thoughts;
It sorteth and indenteth my program.

It RESTOREth my DATA;
It leadeth me in paths of readability for its own sake.

Yea, though I walk through the valley in the shadow of BASIC,
I will fear no GOTO:
for it is state-of-the-art;
its PROCs and its FUNCs they comfort me.

It prepareth an unassigned data space before me in the absence of my values:
It reporteth my errors at source, even if my stack runneth over.

Surely goodness and reliability shall follow me all the days of my life:
and I will dwell in its syntax forever.

Reprinted from ICPUG, Sept/Oct 1986. Collected by Brian Grainger from the 1986 COMAL Standardization meeting. ■

Extra Programs

CAPITOLS

Richard and Todd Shagott have concocted an educational game driver called *can'you'tell'me*. The program uses the file *dat.quiz'menu* which contains the names and data files of each available game. They included games to guess 50 U.S. capitols and 315 world capitols. The data for these games is in *dat.capitols* and *dat.world'cap*. To play the games:

RUN "can'you'tell'me"

The uniqueness of this program is that once it asks the question, it randomly begins to insert letters into the answer, one at a time after a short delay. It decrements the possible number of points available as each letter is inserted. All questions are randomly selected, and then removed from the selection pool only if answered correctly.

In actual play, the <*space*> bar interupts play for the purposes of entering a guess, pausing for a break, or quitting. The games are on *Today Disk #15*.

3D AIRPLANE REVISITED

Today Disk #15 contains updated versions (0.14 and 2.0) of the *airplane* programs by Herbert Denaci from *COMAL Today #13*. They are *1520-3d'airplane* and *chip.airplane*. The 2.0 version first draws the airplane on the hi-res screen. It then checks to see if the program is running on a c128 with Super Chip. If it is, then it draws the airplane on the ultra hi-res (640*200) screen (switch your monitor to 80 column mode to view the plane). ■

Interrupt Package 

by Dick Klingens, Dutch COMAL Users Group

In *COMAL Today #8* Jesse Knight described the use of the **INTERRUPT** command. His article handles only an interrupt by pressing the *<stop>* key. Our package **irq** enhances the COMAL **INTERRUPT** command by letting any of five separate events trigger the **INTERRUPT** command:

- pressing the STOP key
- an internal alarm being given
- there was a sprite/sprite collision
- there was a sprite/data collision
- the countdown timer reaching 0.

The events are detected by the main function in the package called **event**. **Event** is actually an 8 bit register (only bits 0-4 are used).

The events mentioned above have taken place if **event** has the following values (or a combination):

EVENT	bit #	meaning
0	-	nothing until now
1	0	STOP key
2	1	alarm clock
4	2	s/s collision
8	3	s/d collision
16	4	countdown alarm

The alarm clock time can be set by the new command **setalarm(string$)**. The parameter has the same syntax as in **settime** (in the **SYSTEM** package). When the time, set by **settime** and updated by the system, equals the time set by **setalarm**, COMAL calls the **INTERRUPT** procedure.

To detect this, the value of **event** must be tested. This can be done with **BITAND**. After testing the value of **event**, the function is reset to zero by the package (as is done after any test of **event**).

```
USE irq
USE system
settime("0:0:30")
setalarm("7:30:00")
INTERRUPT request
PAGE
LOOP
 PRINT AT 10,10: gettime$
ENDLOOP
//
PROC request
 IF event BITAND 2 THEN
  PRINT "get dressed and go to work")
  bell(255)
 ENDIF
ENDPROC request
```

The command **setcount (integer#)** sets the COMAL timer, where **integer#** is in the interval 1-255. The timer starts counting down to zero once procedure **count(TRUE)** is called. **Setcount(1)** is approximately 1/60 second so the entire countdown is limited to about 4 seconds.

Example:

```
USE irq
USE system
INTERRUPT request
setcount(30); count(TRUE)
off:=TRUE
PAGE
LOOP
 PRINT AT 10,10: gettime$
ENDLOOP
//
PROC request
 IF event BITAND 16 THEN
  IF off THEN
   PRINT AT 10,1: "Time now:",
   ELSE
   PRINT AT 10,1: SPC$(9),
  ENDIF
  off:=NOT off
ENDPROC request
```

More ►

When the timer value equals 0, it calls the **INTERRUPT** procedure and starts again, as can be examined with the example above. This lets **INTERRUPT** be used as a *watchdog*. The **INTERRUPT** procedure can check on something during a lengthy program segment. This may even be the start of parallel processing. The countdown can be stopped by **count(FALSE)**.

Testing sprite collisions is done by the package on machine code level during the IRQ interrupt which occurs every 1/60 of a second. This interrupt calls the **INTERRUPT** procedure in COMAL. The sprite number can be asked with two functions **sscol** and **sdcol** for sprite/sprite and sprite/data collision. The functions act as 8 bit registers, one bit for each sprite. After testing one of these functions, its value is reset to 0:

```
PROC request
 IF event BITAND 4 THEN
  // examine s/s collision
  sp#:=3 // we want to test sprite# 3
  IF sscol BITAND 2^sp# THEN
   //
   // statements
   //
  ENDIF
 ENDIF
ENDPROC request
```

On *Today Disk #15*, there are a few examples using the package **irq**. These files are named *demo/irq1* through *demo/irq4*. *Demo/irq2* is also listed at the end of this article.

[Editor's note: we feel that this package may have widespread applications for powerful COMAL programming. However, the package is not as friendly as it could be. For example, we would like to see separate functions for event and sprite collision detection that do not require the use of BIT functions to interpret them. We have included the source code on Today Disk #15 for those who want to update this package.

Clarification: the package irq use machine code during the hardware irq interrupt which occurs 60 times per second. This code sets flags to be used by the COMAL system. After a COMAL program line is completed, COMAL checks the flags to see if the INTERRUPT procedure should be called. The procedure is not called during the hardware irq interrupt.]

```
USE irq
USE system
INTERRUPT request
PAGE
PRINT "COMAL interrupt-example 2"
PRINT "Demo?  Press <stop>"
PRINT "Quit?  Press spacebar"
TRAP ESC- // <stop> must be disabled
REPEAT
 NULL
UNTIL KEY$=" "
TRAP ESC+ // enable <stop>
CURSOR 20,1
END "End of example"
//
PROC request
 WHILE ESC DO NULL // clear ESC
 IF event BITAND 1 THEN // stop key
  PRINT AT 10,10: "<ESC> was pressed"
  bell(2)
  PRINT AT 10,10: SPC$(17)
 ENDIF
ENDPROC request
```

TYPE QUICK

Norbert Bakker sent this typing game from Holland. Try to type the words as they scroll onto the screen from the right before they crash on the left. There are 26 levels from hunt and peck to the fastest touch typing. Choose a level to start from and automatically get promoted as you sucessfully type nine new words. The COMAL keywords have been selected as the words to type, but you can substitute a new list by changing **DATA** statements.

Rearrange Programs

by Doug Drake

This may not be news to every COMAL 2.0 programmer, but it was new to me. It has improved the organization of my programs immensely.

One thing that cannot be done easily by the COMAL screen editor is to rearrange blocks of program code. As I develop a program, I tend to bring in each of my standard utility procedures as I need them to test the most recent program module. Thus my procedures and functions tended to end up in a jumble - first the menu, then a couple of utility routines, then the first program module, more utilities, another program module, and so on. Rearranging these program components can be a tedious task.

Nearly every word processor, on the other hand, has simple commands for rearranging blocks of text. Once the line numbers are removed, your program is just text. Why not use a word processor, I thought, to do the work?

After about 15 minutes of experimentation, I figured out how to accomplish this task. The specific steps you'll need to follow will vary depending on your particular word processor, but the general idea remains the same.

First, be certain that your program is in working order. Any errors before the program is rearranged may be magnified afterwards.

Second, from COMAL 2.0 with your program in memory, issue this command:

DISPLAY "filename"

If your word processor uses true ASCII files instead of PETASCII, add **"/a+"** to the end of the filename (ie, **"filename/a+"**). If it uses screen codes, just write a PETASCII file for now. This will write your program to the file specified above with the appropriate line indentation, but without line numbers.

If your word processor uses screen codes, you will now need to run whatever conversion program you use for sequential files - *speed'to'seq* from *Today Disk #9*, for instance. You will need to run the reverse conversion program later, before you re-enter the file as a COMAL program.

Now to the third step. Run your word processor and load the file which was written above. Before you do anything else, insert a line at the top of the program and type:

AUTO

An alternative approach is to have **AUTO** as a comment for your first program line, then simply delete the comment notation at this step. Note, the first line must be **AUTO**.

You are now ready to do your text rearranging. Be careful not to destroy the program structure here. You're simply moving entire procedures or functions around. If you try to change the program itself, the results could be disasterous. The program indentations help you locate the modules. When you are done, save the file and go back to COMAL. Convert from screen codes back to PETASCII if necessary.

Now for the fun part. From the COMAL editor, issue the command:

SELECT INPUT "filename"

More ►

just like you would for a batch file. Specify ASCII translation as part of the filename if necessary (ie, "**filename/a+**"). The first line of the file, **AUTO**, will automatically number all program lines as they come in from the file. Your program lines will be numbered sequentially and entered into memory. When the operation is complete, you'll see the last program line near the bottom of the screen, followed by a line number and the blinking cursor. Hit the *<stop>* key and save your rearranged program. You may have to hit the *<return>* key once first to enter the last program line. That's all there is to it!

There are some potential pitfalls to this process. The biggest is that any error encountered will not merely interrupt the process, but effectively terminate it. That's because the program lines keep streaming in, but no longer make any sense. To abort this process, hit *<run/stop>* and *<restore>* together.

However, if your program was OK before this procedure, it should still be OK for re-entry. The only exception to that statement which I've found was in the case of an extremely long program line. Since **DISPLAY** maintains program indentation, a line which fit in 80 columns on initial entry can conceivably be too long with indentation included. Make a quick scan of the file while in the word processor. Look for lines over 74 characters; to avoid this pitfall, just delete the indenting spaces to make the line shorter. The program indentation need not be correct for re-entry into the COMAL editor.

A couple of other things to watch out for: if your word processor places soft carriage returns at the end of each line to allow you to change margins or re-format, **don't do it**. That will turn your program into garbage. Second, if you get involved in ASCII translation, you may find that a couple of characters don't translate properly. For instance, using *Wordwriter 128* I find it necessary to use **CHANGE** to get the left and right brackets back. This is a simple matter in COMAL 2.0. I just make the changes right before I save the program.

As a result of this procedure, my program listings now make a lot more sense. I'm able to group my procedures and functions so that utilities are grouped together, program modules follow the order of the menu, and so on.

The procedure works especially well on the C128. Some C64 word processors don't have enough memory for long programs. With the proper software, the program can be viewed in 80-column mode. Program structure is more difficult to follow on the 40-column screen, but everything still works. Using Easy Script for instance, I'd suggest setting the line length to 80. It's only the left-hand portion of each line you need to see.

I have tested this process on both of the word processors mentioned above, as well as *Speedscript*. I think you should be able to find a way to make it work with just about any wordprocessor, one way or another.

[Editor's note: We sometimes use this method to insure that COMAL 2.0 programs for the magazine have survived PaperClip editing. This method does not work in COMAL 0.14 which lacks DISPLAY and SELECT INPUT. However, programs could be written to remove line numbers from listed files and put them back after the files have been rearranged by a word processor.] ▣

TRON - the New Trace

by Richard Bain

Commodore COMAL 2.0 has a powerful, yet little used debugging utility built in. It is the **TRACE** command. After a program stops, you can type **TRACE** to see how the computer got to the line it stopped at. What it actually tells you is each procedure or function you were inside at the moment the program stopped, and the line they were called from. **TRACE** is useful for debugging recursive procedures and programs where several procedures call other procedures. **TRACE** is not useful in determining program flow. It won't tell that you entered a procedure if you also exited it. **TRACE** is no help at all if the program doesn't have procedures.

Another powerful, yet little used feature is the **INTERRUPT** command. Under a few select conditions, COMAL jumps to the specified **INTERRUPT** procedure after completing a statement. This can be used for a new type of trace command: **tron**.

A useful debugging method on some computer systems involves printing each statement's line number as it executes. The **INTERRUPT** command lets this be done from COMAL by calling a procedure after each line is executed. The procedure then prints the number of the line which called it.

Demo/tron on *Today Disk #15* shows how to use **INTERRUPT** to follow program flow from line to line. To use this feature in other programs, first merge the four procedures on the next page into your program. Start the program with a call to **interrupt'init**. Insert **tron** before the lines you want traced and **troff** when the section you want traced ends. Also include this line in all your **CLOSED** procedures:

```
IMPORT print'line'number,tron,troff
```

When COMAL calls a procedure, it keeps track of where it came from in an organized manner. The vector at $10 points to the start of the stack entry for the current procedure, the fifth and sixth bytes of which point to the address of the line where the procedure was called. The first two bytes of this line contain its line number; this is what you are looking for. Procedure **print'line'number** finds this information.

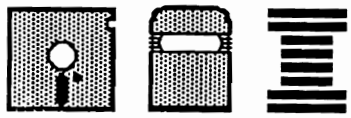
For our **tron** command to print line numbers as they are executed, we must first initialize the **INTERRUPT** procedure. **Interrupt'init** pokes a short machine language routine into unused memory beginning at $c86a. The routine sets bit 2 of EXCINF ($4d) which tells COMAL to call the **INTERRUPT** procedure if **INTERRUPT** <*procname*> is active. Then **interrupt'init** changes the vector USRQVC ($c7e2) to point to the machine language routine. Finally, **interrupt'init** sets bit 5 of EXCINF telling COMAL to execute the machine code pointed to by USRQVC after executing each COMAL program line. Nothing this procedure does is specific to the **tron** command. It only tells COMAL to execute the procedure specified by **INTERRUPT** <*procname*> between each COMAL program line. If **INTERRUPT** is issued without a procedure name, then no procedure will be called between program lines. See *Using the Interrupt Command, COMAL Today #8*, page 62 for more details.

Now you have a procedure which knows where it was called from, and a way to call that procedure after each line. What is still needed is a way to turn this feature on and off. Why print 100 line numbers before you get to the part of the program that is causing you trouble? Procedures **tron** and **troff** tell COMAL whether or not to use this feature.

More ►

```
// delete "demo/trace"
// save  "demo/trace"
//  by Richard Bain
//
PAGE
PRINT "Demonstration of a TRACE"
PRINT "function for COMAL 2.0."
PRINT
PRINT "Press space bar to view"
PRINT "line numbers as they"
PRINT "execute. Press space bar"
PRINT "again to stop listing"
PRINT "lines."
PRINT
PRINT "Press STOP and type LIST"
PRINT "to see the program."
PRINT
//
interrupt'init
state:=FALSE
LOOP
  IF KEY$=" " THEN
    state:=NOT state
    IF state THEN
      tron
    ELSE
      troff
    ENDIF
  ENDIF
ENDLOOP
//
PROC interrupt'init CLOSED
  FOR address:=$c86a TO $c870 DO
    READ byte
    POKE address,byte
  ENDFOR address
  POKE $c7e2,$6a
  POKE $c7e3,$c8
  POKE ($4d),PEEK($4d) BITOR $20
  //
  DATA $a5,$4d // lda $4d  - excinf
  DATA $09,$04 // ora #$04 - srq
  DATA $85,$4d // sta $4d  - excinf
  DATA $60 //     rts
ENDPROC interrupt'init
//
PROC print'line'number CLOSED
  pt:=PEEK($10)+256*PEEK($11)
  la:=PEEK(pt+4)+256*PEEK(pt+5)
  ln:=256*PEEK(la)+PEEK(la+1)-10000
  // select "lp:"
  PRINT USING "####": ln;
  // select "ds:"
ENDPROC print'line'number
//
PROC tron
  INTERRUPT print'line'number
ENDPROC tron
//
PROC troff
  INTERRUPT
ENDPROC troff
```

Pitfall



by Dick Klingens and Joe Visser
Dutch COMAL Users Group

1. Intro

The title of this article sounds somewhat negative, but that is not entirely wrong. By its complexity COMAL sometimes offers the programmer strange effects.

2. REF parameters

Many high level programming languages support parameter transfer into procedures in one way or another. So does COMAL. As with Pascal, there are two parameter types, value and reference. Look at the following procedure heading:

PROC one(x, REF y) CLOSED

This procedure has parameters of each type:

x is a value parameter
y is a referenced parameter.

The difference between these types is, that after a call of the procedure **one** a *value* is transferred to x and a *variable* to y (including the variable's value). The variables x and y both die immediately after leaving the **CLOSED** procedure, but before that, y returns its value to the actual parameter in the procedure call.

The call:

EXEC one(4,7)

is invalid, because the second parameter, *7*, is not a variable. (Note: the keyword **EXEC** is optional and may be omitted).

However, after the call:

EXEC one(4,hi)

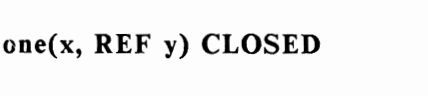
the variable **hi** is assigned the same value as y just before y stopped existing at the end of the **one** procedure.

3. Inside COMAL

COMAL sees its variables consisting of three parts:

1. its name
2. a pointer to its value
3. its value

The pointer part points at the value, like this:

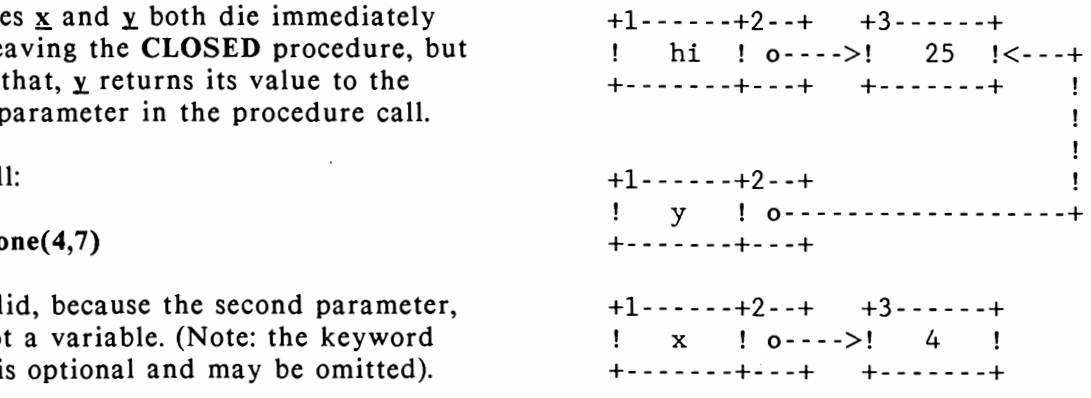


saying that **hi** has the value 25.

Suppose that the procedure **one** is called with:

EXEC one(4,hi)

Now we have



Two new variables are created temporarily. x has its own value part; the pointer part

More ►

of y points at the value part of hi. Changing y means changing hi too. This is why the value of hi equals the value of y when we leave the procedure.

4. Consequences

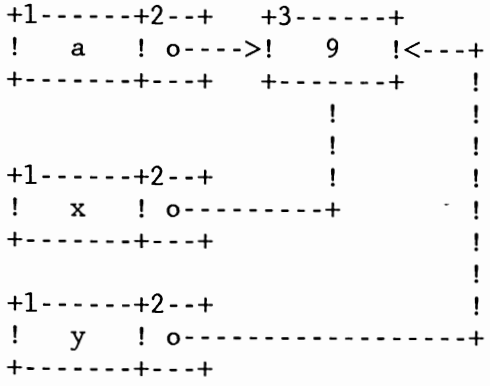
Look at the procedure:

```
PROC two(REF x, REF y) CLOSED
  x:=1
  PRINT y
ENDPROC two
```

Start with a:=9. The call:

```
EXEC two(a,a)
```

gives the output *1*, **not** *9*, as perhaps expected. We didn't change the value of y, did we? We only changed x. COMAL changed y. We illustrate with a diagram:



Both x and y are pointing at the same value part (that of a). Because of this it's not surprising that x:=1 has some influence on the value of y.

You will say that in programming practice there never will be a situation in which two **REF** parameters are equal. This is less true, than it looks at first sight. We'll explain why.

Examine the procedure swap, which swaps the values of two variables without a third one.

```
PROC swap (REF x, REF y) CLOSED
  x := x + y
  y := x - y
  x := x - y
ENDPROC swap

a:=9; b:=10
EXEC swap(a,b)
PRINT a; b
```

10 9 is the output.

```
a:=9
EXEC swap(a,a)
PRINT a
```

the output is *0*.

How come? Draw a diagram yourself or do some minor mathematics:

```
a := a + a
a := a - a
a := a - a
```

No one will program a statement such as:

```
EXEC swap(a,a)
```

but it is possible to transfer elements of an array into the procedure swap and call

```
EXEC swap( b(i), b(j))
```

You will find some unexpected effects when i=j; then the cause is not so obvious.

Further reference:

Scope Rules, COMAL Today #14, page 60
Pass an Array as a Parameter, COMAL Today #4, page 48
Parameters, COMAL Today #2, page 32 ■

Super Chip Notes

[When you read this, Super Chip on Disk should be available. It should include all the packages and commands of Super Chip except autostart. The RAM version can be linked to programs or the ROMMED version can remain in memory while the computer remains on. The packages use nearly all the memory under the COMAL cartridge so it is unlikely they will work with any other disk loaded packages (even ones which work with Super Chip).]

This article uses **quicksort**, **host$**, **turbo**, **hidescreen**, **showscreen**, and **lowercase$**, to alphabetize a list of data which may contain some capital letters. This makes sorting difficult because "A" > "z" even though "a" < "z". Another problem comes from trying to make the sorting process fast. The c128 can run faster than the c64, so why not take advantage of its speed. The following program converts data to lowercase for easy sorting, and checks the computer type to optimize speed.

```
0010 USE strings
0020 max:=10
0030 DIM name$(max) OF 15
0040 read'array(max,name$())
0050 FOR x:=1 TO max DO name$(x):=lowercase$(name$(x))
0060 speed'up(TRUE)
0070 quicksort(name$(),1,max)
0080 speed'up(FALSE)
0090 PAGE
0100 FOR x:=1 TO max DO PRINT name$(x)
0110 //
0120 PROC read'array(REF max,REF name$()) CLOSED
0130   FOR x:=1 TO max DO
0140     INPUT "type someone's name: ": name$(x)
0150   ENDFOR x
0160 ENDPROC read'array
0170 //
0180 PROC speed'up(hurry) CLOSED
0190   USE system2
0200   IF host$="c128" THEN
0210     USE c128
0220     turbo(hurry)
0230   ELIF hurry THEN
0240     hidescreen
0250   ELSE
0260     showscreen
0270   ENDIF
0280 ENDPROC speed'up
```

Line 10 is needed to activate the **lowercase$** and **quicksort** commands. Super Chip owners must issue the proper **USE** command before any of the added commands can be called.

Lines 20 - 40 are used to obtain the names to be sorted. Line 40 has been made into a procedure call to make it easier for programmers to change the input routine. The names may be **READ** from **DATA** statements or a disk file if necessary.

Line 50 is used to convert every character of the names in the array to lowercase (Sam becomes sam). This is important for sorting purposes if you don't have control over how the data is formatted. However, the change is permanent. It may not be easy to restore some of the letters back to capitals.

Line 60 calls a procedure to speed up the computer. It will be discussed in detail later.

Line 70 calls the **quicksort** procedure to sort the array. Note that the three parameters are the string array to be sorted, the number of the first element of the array, and the number of the last element of the array. The numbers can be changed if the entire array does not need to be sorted.

Line 80 is used to return the computer to its normal state with the 40 column screen showing. Lines 90 - 100 clear the screen and print the sorted list of names.

Lines 110 - 160 are the procedure called from line 40 to get the names. This procedure can be changed as needed to get data from other sources.

Lines 170 - 280 are a procedure which you

More ►

may want to add to other programs. It is used to speed up parts of a program which might otherwise be too slow to be interesting. Programs with tedious math routines should use this procedure. Warning: do not try to use this method to speed up the computer during file access.

Line 180 is the procedure header. The parameter, **hurry** should be given the value **TRUE** or **FALSE** from the calling statement (lines 60 and 80). Line 190 activates the **host$**, **showscreen**, and **hidescreen** commands. Line 200 tests for which computer the program is running on, a c64 or a c128.

If the computer is a c128, the next two lines will be executed. The **USE c128** command of line 210 should almost always be put inside an **IF** structure or a **TRAP - HANDLER** structure. Otherwise, it could cause the program to **STOP** with an error on a c64. Line 220 will either blank the 40 column screen on the c128 and set the fastmode, or restore the 40 column screen and normal speed. It depends if the value of **hurry** has been set to **TRUE** for fast or **FALSE** for normal.

If the computer is a c64, the computer will skip lines 210 - 220 and continue with line 230. The screen will be blanked in line 240 or restored in line 260 depending on the value of **hurry**.

If you are writing programs for personal use, you might think it is silly to check what kind of computer the program is running on. After all, if you only have a c128, why add extra code to let the program run on a c64? Worse yet, if you just have a c64, how will you know if the extra commands even work on a computer you can't test them on.

The second question is easier to answer. It takes a very good programmer to write code that works the first time untested. If you don't have the hardware needed to test a program, it probably isn't worth the effort to try to write software for it.

The first question is harder to answer. The purpose of the above program is to quickly sort a list of names. Testing for a c128 can make a significant difference in speed. However, the speed gained from the c128 fastmode is not a good reason to prevent the program from working on a c64. Even if the program is for personal use, you may want to show the program to a friend. It will be a dissapointment if the program won't run. A few programs may need the advantages the c128 offers COMAL through Super Chip. They might not be usable on the more limited c64. This is an example of a program that takes advantage of the c128, but not at the expense of the c64 users.

SUPER CHIP ON DISK

Now all the owners of the original beige 2.0 cartridge can use Super Chip. **PLUS** - Super Chip owners now can share their programs with **any** COMAL 2.0 cartridge user! Just **LINK** the Super Chip packages to your program! We will copyright the disk and the package modules, but the disk will not be copy protected. You may make a backup of the disk for your own archival use, but you will not be permitted to make copies for friends or user group. It will be OK to give copies of programs with Super Chip LINKed to it, but not the packages alone. The disk is available to subcribers for $19.95. Get **both** the chip and the disk **combo** and deduct $6. ■

Bird Data Base

by Bob Hoerter

I developed a database for information my wife and I collect while pursuing our hobby of bird watching. It is on *Today Disk #15* with a sample data file and its subfiles. I am not totally satisfied with the system, but it is to the point where other COMAL users may find it of interest (even if they are not bird watchers). Many parts of the program were borrowed from, or inspired by, earlier articles published in *COMAL Today*. Captain COMAL's *Telephone Number Database* on page 45 of *COMAL Today #10* and *shell'sort* on *Today Disk #6* were the starting point for this system.

I chose the free form system used in the telephone directory system, rather than a system which depended on random access files, because I wanted to add data over a long period of time. The main file could become quite large and sequential files read faster than random files when doing searches. The main data file is used as the source file from which many subfiles can be created to speed up information retrevial. As the data grows, it will exceed the computer's memory, so I did not use an array based system.

All information is entered as part of one 66 character string. I used one variable to save disk space and simplify sorting.

One of the nice things I discovered is that by using substring notation in **INPUT** statments, you can get neatly formatted data. For example:

```
INPUT "First Name":name$(1:10)
INPUT "Last Name":name$(11:25)
```

makes name$ exactly 25 characters. Any excess in **INPUT** is dropped and any shortage is padded with spaces.

The 2.0 string function capability enables me to enter a three character code and let a **FUNC** expand it to as many as 20 characters. This speeds up data entry and saves disk space. The code to be entered and its value for the function to **RETURN** are stored as data statements.

Subfiles of the main data file are used for specific infomation which you may want to access quickly, without working with the main database file. These subfiles are all prefixed with *bf* to identify them as belonging to this system. Should you try to read a file that has not been written, you will be shown a directory of the *bf* files on the disk. These subfiles are read into memory as string arrays so that they can easily be resorted by any field the user chooses (example: date or location). Using shell sort, with only one string per record, does this quickly and easily because it compares substrings, but swaps entire strings.

Should a mistake be made in entering data to the main file, a program in the system can either correct the mistake or delete the incorrect record. Though this program is menu driven, it is reccomended that you practice on a sample file before working on the real one. These changes to the database file are made by reading the file, and writing a new *temp* file which may be reviewed before the changes become permanent. If you are satisfied with the *temp* file, the original file is deleted and the *temp* file renamed.

One final note. The subfiles are made using the dreaded @ (save and replace). The COMAL cartridge may have solved this problem in that I have never experienced a problem with this while using COMAL. ■

Matrix Use

by Robert Ross

Pkg.matrix is a machine language implementation of **PROC matmult** given in *matpac.l* on *Today Disk #2*. Each element of the answer array, **c(m,o)**, is given the value of the sum of the **n** products of the **n** corresponding elements of two source arrays: **a(m,n)** and **b(n,o)**. *Pkg.matrix* is on *Today Disk #15*.

```
PROC matmult(REF a(,),REF b(,),REF c(,),m#,n#,o#)
 FOR j#:=1 TO m# DO
  FOR k#:=1 TO o# DO
   temp:= 0
   FOR l#:=1 TO n# DO
    temp:= temp+a(j#,l#)*b(l#,k#)
   ENDFOR l#
   c(j#,k#):=temp
  ENDFOR k#
 ENDFOR j#
ENDPROC matmult
```

This package saves time in the indexing of array elements only (the existing floating point arithmetic is used and takes the same amount of time). It is fair to say that the package is three to four times faster than the COMAL procedure it replaces. The exact time in a comparison test depends on the size of the arrays and the actual values of the elements.

After linking *pkg.matrix* and commanding USE **matrix**, the procedure is called with **matmult(a(,),b(,),c(,))**. **A**, **b**, and **c** are the names of the actual arrays to be passed by reference. COMAL generates an error at the procedure call if any of the arrays are not two dimensional. The procedure will generate a non-trappable runtime error ("error" 64) if the dimensioned ranges of any of the three indices do not match. For example, if **m** for array **a** is 1:10 and **m** for array **c** is 1:11, error 64 is reported; however, 1:10 will match 2:11 since each is ten elements. The system floating point routines will report error 2 "overflow" if it occurs. ■

How to Type in COMAL Programs

Line numbers are required for your benefit in editing a program. Thus line numbers often are omitted when listing a COMAL program. It is up to YOU to provide the line numbers. Of course, COMAL can do it for you. Follow these steps to enter a COMAL program:

1) Enter command: NEW
2) Enter command: AUTO
3) Type in the program
4) When done: COMAL 0.14: Hit <return> key twice
   COMAL 2.0: Hit <STOP> key

While entering a program, use unshifted letters. If letters are capitalized in the listing it does not mean to use SHIFT with those letters. They are capitalized merely to be easy to read. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures. You DO have to type a space between keywords in the program.

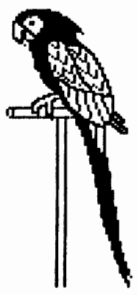
Long program lines: If a complete program line will not fit on one line, we will continue it onto the next line and add //wrap at the end. You must type it as one continuous line. Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz 0123456789'] [<backslash> _

The <left arrow> key in the upper left corner of the keyboard is valid. COMAL 2.0 converts it into an underline. If you see an underline in a program listing, type the <left arrow> key. The C64 and C128 computers use a <british pound> in place of the <backslash>. ■

Parrot -Expert System News

THE PARROT



PARROT is a vehicle for communication, established to inform and support the users of the PROTOTYPE DELTA Expert System Shell.

Address: 'PARROT'
65 East 24th Street
San Angelo, Texas 76903

Mike's Notes: September 86

Thanks to Len, David and Rick the PROTO-D expert system shell is now available thru the COMAL Users Group as shareware! This means I reserve rights to the program and you get to decide if you like it well enough to send me some small token of your appreciation. I hope it earns your appreciation!

I want to start a SIG for writers of PROTO-D knowlege bases. Documentation for the programs is rather inadequate and this is the vehicle which will be your best source of documentation for the system.

Funding is scarce so before I make a total commitment to doing a news letter I need to poll interested persons. Please write 'PARROT', 65 East 24th St., San Angelo, TX, 76903, expressing your interest. Send an SASE and 10 cents for the next issue.

PROTO-D EDITOR MADE PUBLIC DOMAIN!!

If you have looked at the listings of the PROTO-D system you may have noticed the Editor program does not show a copy write. This was an error made during editing (not the fault of the author). Since the editor needs to be improved and as others may wish to use the editor in their public domain programs, I place the PROTO-D editor program into the public domain. This allows any programmer to improve the editor.

I am starting a fall semester. COBOL and IBM Assembler and Digital Electronics, will probably keep me from getting to that improvement before Christmas break.

Michael Gerstenberg
2 Sept 1986

PROTO-D IMPROVEMENTS

PROTO-D could be improved many different ways. One of the first improvements which PROTO-D will see is the elimination of the requirement to enter a rule when entering new answers with the WRITER program. It will probably be necessary to divide the WRITER and READER into 2 seperate programs to compensate for a partial system. Still it will allow one to write either a pattern matching system or a rule based system or both. I think this will improve the flexibility of PROTO-D. I hope that the added flexibility encourages the writing of more expert systems.

When that project is finished I will begin converting the READER only to COMAL .14! That will open wide the door to those who wish to see a large (paying) market for their knowlege bases.

Our Responibility,

I have chosen the parrot as our mascot because Prototype Delta is just such a bird! It will repeat only what it has been taught to say. As writers of the PROTO-D systems, we face a challenge and an opportunity. We must develop a system of programs that solves problems which have never before been solved by computers. We have a responsibility to the end users of that system to produce an accurate, reliable, and comprehensive system. With the PROTOTYPE DELTA WRITER we also have the POWER to produce such a system. In short we have a responsibility to strive for excellence.

'PARROT'

Authors Wanted!!

Knowlege Engineers are NOT a dime a dozen! Who could or would expect an expert to impart hard earned wisdom and experience to a computer program and then freely give it away? Certainly not !! If you are writing a knowlege base which runs under the PROTOTYPE DELTA system and wish to be compensated for your work, send a letter and SASE requesting a statement of non-disclosure to 'PARROT' at the address to the left. We will forward to you a statement (legally binding) that we will not publish or reproduce your work without your written permission. When you have received this statement you should then send your finished and ready to publish knowlege base to the same address. We will review your work. If it meets our criteria as shown in our responsibility statement, we will endorse and make available your knowlege base to other users of the POTOTYPE DELTA system. Knowlege bases will be priced at between $5.00 and $20.00 each. Authors will be paid 20% of gross for each copy sold.

Copy Protection:

It is our feeling that copy protection violates the users right to free personal backup of software purchased for his own use. Since PROTOTYPE DELTA is not protected (it is copywrited) we do not feel and do not authorize any party to use PROTOTYPE DELTA to produce copy protected or commercial software and we will not publish or endorse protected knowlege bases.

COPYWRITING:

If we opt to publish your knowlege base you will maintain the copywrite to your knowlege base and we will provide a written contract to ~~she~~ you requesting sole license to publish your work for as long as you continue to accept payment of royalties from Homegrown Software. This allows either party to terminate the agreement at any time.

Thats the deal, not bad really, think about it and then get to work! There is a serious lack of finished knowlege bases around here. First come first served!

THINK STRUCTURE!

Order Form (MORE ITEMS ON OTHER SIDE)

Name:_____ SUBSCRIBER NUMBER:_____ Nov 1986
(required for reduced prices-except new subs)
Street:_____ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:_____ VISA / MasterCard print card#/exp date:

VISA/MC #:_____exp date:_____Signature:_____

Qnty Price List/Subscriber price-Item Description (all disks Commodore 1541 format) Prices subject to change

BOOKS:

(Canada & APO / FPO & First Class shipping add $1 more per book)

[ ] _____ **$6.95/$4.95 COMAL Today--The INDEX, Kevin Quiggle, 56 pages, 4,848 entries (ship $2)**
[ ] _____ **add $7.95 for matching disk! Contains SEQ text files of entries! Plus some 2.0 programs**
(COMAL Today issues 1-12, articles, notes, authors -- indexed!)
[ ] _____ **$19.95/$17.95 Introduction to Computer Programming, J William Leary (272 pages)-(ship add $3)**
[ ] _____ **$6.95/$4.95 Answer Book to Introduction to Computer Programming (64 pages)-(ship add $1)**
(Beginners text book for American students, spiral bound for easy use)
[ ] _____ $18.95/$16.95 COMAL Handbook, 2nd Edition, Len Lindsay (479 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Detailed Reference book to COMAL 0.14 and 2.0)
[ ] _____ $19.95/$17.95 Foundations With COMAL, 2nd Edition, John Kelly (363 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Beginners text book, Jr/Sr High level, by Irish professor)
[ ] _____ $28.95/$26.95 Structured Programming With COMAL, Roy Atherton (266 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Intermediate text book, stressing modular programming, High School level, by British professor)
[ ] _____ $20.95/$18.95 Beginning COMAL, Borge Christensen (333 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Beginners text book, Elementary school level, by Danish professor, founder of COMAL)
[ ] _____ $6.95/$4.95 COMAL From A To Z, Borge Christensen (64 pages)-(shipping add $2)
(Mini reference book for COMAL 0.14, including its graphics and sprites)
[ ] _____ $14.95/$12.95 Captain COMAL Gets Organized, Len Lindsay (102 pages, with disk)-(ship $2)
(Application tutorial to illustrate benefits of modular programming)
[ ] _____ $6.95/$4.95 COMAL Workbook, Gordon Shigley (69 pages)-(shipping add $2)
(Beginners level - perfect companion to the Tutorial Disk)
[ ] _____ $14.95/$12.95 Graphics Primer, Mindy Skelton (84 pages, with disk)-(shipping add $2)
(Beginners level tutorial to COMAL 0.14 graphics and sprites)
[ ] _____ $6.95/$4.95 Cartridge Graphics & Sound, Friends of Captain COMAL (64 pages)-(ship add $2)
(Mini reference book to all built in 2.0 cartridge package commands)
[ ] _____ $19.95/$17.95 COMAL 2.0 Packages, Jesse Knight (108 pages, with disk)-(shipping add $2)
(Advanced. For ML programmers- how to write your own packages- with C64sym & supermon)
[ ] _____ $19.95/$17.95 Packages Library, David Stidolph (76 pages, with disk)-(shipping add $2)
(Intermediate. 17 example packages, most with source code on disk- Smooth Scroll Editor too)
[ ] _____ $21.95/$19.95 Cartridge Tutorial Binder, Frank Bason, Leo Hojsholt (320 pgs, with disk)(ship $3)
(Beginners manual to everything in the COMAL 2.0 cartridge)

OTHER:

[ ] _____ OTHER: _____
[ ] _____ **$10.95 Custom white plastic molded case - holds 5 disks and 2 small books**
[ ] _____ **$5.95 Special white plastic folder - holds 1 disk and 1 small book**
[ ] _____ $9.95/5.95 COMALite T-Shirt in Royal Blue - Circle what adult size: S M L XL
[ ] _____ $3.95/$2.95 Keyboard Overlay for C64 COMAL 0.14 - Cheatsheet (shipping add $1)

[ ] _____ **UPS BLUE** Shipping add $3 extra per book, $1 extra per disk (USA only, $6 minimum)
(overseas shipping is extra)
Total_____ + _____ Shipping = US$_____ **Total Paid** (WI add 5% sales tax)-(shipping $2 minimum)

USE OTHER SIDE FOR TOTAL if order includes items on that side

Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Order Form (MORE ITEMS ON OTHER SIDE)

Name:_____ SUBSCRIBER NUMBER:_____ Nov 1986
(required for reduced prices-except new subs)
Street:_____ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:_____ VISA / MasterCard print card#/exp date:

VISA/MC #:_____exp date:_____Signature:_____

Qnty Price List/Subscriber price-Item Description *(two disks may be supplied as one double sided disk)*
Prices subject to change *(all disks except IBM PC COMAL system are Commodore 1541 format)*

SUBSCRIPTIONS

[ ] _____ **COMAL Today newsletter-> How many? _____ Start at: [_] renew [_] current [_] Issue#_____**
(each issue is 80 pages of articles, notes, tips, and program listings)
($18.95 for 6; $26.95 for 10 issues; >>> Canada add $1 per issue; >>> overseas add $5 per issue)
[ ] _____ $3.95/**$1.50** COMAL Today backissue: circle #s wanted-> 5 6 7 8 9 -(ship add $1 each)
[ ] _____ $3.95/**$2.95** COMAL Today backissue: circle #s wanted-> 10 11 12 13 14 -(ship add $1 each)
[ ] _____ $16.95/**$14.95** COMAL Yesterday, first 4 issues of COMAL Today, spiral bound (ship add $3)
[ ] _____ **Today Disk subscription-> How many (at least 6)? _____ Start with disk#_____ or [_] renew**
(each disk contains most programs from COMAL Today - plus more! Double sided since #6)
($35.95 for 6; $55.95 for 10 disks---add $5 per disk after first 6---**(no shipping charge)**
[ ] _____ $14.95/**$9.75** Today Disk-**Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15**
(Disk subscriptions must be 6 or more disks - single Today Disks are available - use above line)

SYSTEMS:

[ ] _____ $138.95/**$128.90** Deluxe Cartridge Pak (with the black COMAL 2.0 cartridge)-(shipping add $5)
(New Pak also includes: two manuals, a disk, and Super Chip)
[ ] _____ **$88.95/$85.95** Black COMAL 2.0 Cartridge, Plain (no books/no disk)-(shipping add $2)
[ ] _____ **$29.95/$24.95 Super Chip (for black COMAL 2.0 cartridge) with C128 support**-(ship $1)
[ ] _____ **$5/$5** Super Chip installation fee. Done for no charge if cartridge purchased at same time.
[ ] _____ **$21.95/$19.95 Super Chip on Disk. The Super Chip added commands now can be disk loaded.**
[ ] _____ **$49.95/$38.90 Super Chip Combo - both the CHIP and the DISK**
[ ] _____ **$29.95/27.95 COMAL 0.14 Starter Kit (5 disks, 2 books, 6 newsletters, more)-(ship add $4)**
(includes COMAL From A to Z book, Demo, Tutorial, and Best Of disks)
[ ] _____ $395/**$395** IBM Denmark PC COMAL 2.0 (Danish manual/COMAL Handbook)-(shipping add $5)
(system is in English, works on most IBM PC Compatibles with at least 256K)

DISKS:

[ ] _____ $14.95/**$9.75** Read & Run disk (COMAL 2.0 only)
[ ] _____ $14.95/**$9.75** Shareware disk (COMAL 2.0 only) -- Buy #1 -- Get #2 FREE.
[ ] _____ $14.95/**$9.75** Best of COMAL 0.14 from 1984 (new version - single side of disk)
[ ] _____ $14.95/**$9.75** Auto RUN Demo and Tutorial Disk (perfect with COMAL Workbook)
[ ] _____ $14.95/**$9.75** Bricks Tutorials (2 sided beginners disk - an expanded Tutorial disk)
[ ] _____ **$14.95/$9.75 Utility Disk for COMAL 0.14 - Circle one wanted: 1 2** (#2 includes COMAL Quick)
[ ] _____ $10/**$9.75** User Group Disks **0.14** - Circle disks wanted: **1 2 3 4 5 6 7 8 9 10 12**
[ ] _____ $10/**$9.75** User Group Disks **2.0** - Circle disks wanted: **11 13**
[ ] _____ $29.95/**$14.95** Set of all Cartridge Demo Disks (includes #1 #2 #3 & #4)
[ ] _____ $14.95/**$9.75** Single Cartridge Demo Disk, Circle one wanted --> **1 2 3 4**
[ ] _____ $14.95/**$9.75** Slide Show Picture Disks - both #1 & #2 - (HRG type pictures & 0.14 system)
[ ] _____ $14.95/**$9.75** Games Disk #1 (for 0.14 & 2.0)
[ ] _____ $14.95/**$9.75** Typing Disk (for 2.0 only)
[ ] _____ $14.95/**$9.75** Modem Disk (for 0.14 & 2.0) - programs on disk may change frequently
[ ] _____ $14.95/**$9.75** Font Disk (for 0.14 & 2.0) - dozens of fonts (compatible with PaperClip too!)
[ ] _____ $94.05/**$89.95** 19 Disk Set (about 1000 programs for COMAL 0.14)-(shipping add $3)

=====
Total_____ + _____ Shipping (this side) (Canada & First Class add $1 extra per book)
Total_____ + _____ Shipping (other side)
===== + ===== *(overseas shipping is extra)*
Total_____ + _____ = US$_____ **Total Paid** (WI add 5% sales tax)-(shipping $2 minimum)

Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Midnite Software Gazette

P.O. Box 1747
Champaign Illinois, 61821
Phone (217) 356–1885

Hello and welcome to the Midnite Software Gazette! We are the oldest independent magazine in North America for users of Commodore brand computers.

The Midnite Software Gazette was born to provide a forum for reviews of hardware and software for Commodore computers. Before becoming a commercial magazine, Midnite was published by Jim Strasma and myself as the newsletter for the Central Illinois Pet User's Group. Since that time, over seven years ago, we have published over 1600 reviews. And we are still publishing the same hard hitting reviews that the you need when looking for hardware and software.

The material in the Midnite Software Gazette is written by users--by the experts recognized throughout the Commodore world, and by the home hobbyist who has something to say. Literally hundreds of names have by-lined our reviews and articles, and we always invite you to contribute as well. We need your input as well as your support.

Sincerely,

Jim Oldfield, Jr., Editor-in-Chief.

MIDNITE SOFTWARE GAZETTE SUBSCRIPTION FORM

F. Name:_____ L. Name:_____

Street:_____ Apt. #:_____

City:_____ State:_____ ZIP:_____

One Year (12 Issue) Subscription: $ 23.00, $43.00 Air Mail. Payments are accepted in United States funds as check or money order. MasterCard and Visa are also accepted.
Pay by: Check:_____ Money Order:_____ MstrCrd:_____ VISA:_____

Credit Card Number:_____

Exp. Date:_____ Signature:_____

Back issues (12-15, 17-21, 23-27) available at half of cover price

---

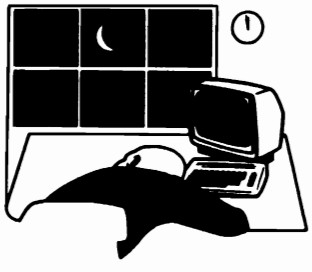
MIDNITE SOFTWARE GAZETTE REVIEW FORM

PRODUCT:_____ Author:_____
Price:_____ Media:_____ Type/Application:_____
For computer:_____ Company:_____
Required Equip:_____ Optional Equip:_____
Protected? ___ How?_____ Warranty?_____
Similar to:_____ Compatible with:_____

In 250 to 500 words, describe the program, tell what you liked, what you did not like, what standard features are/are not implemented, and who should buy it. Then, considering how well it works, its price, and compatibility, state whether the product is NOT RECOMMENDED, AVERAGE, RECOMMENDED, or HIGHLY RECOMMENDED. Include your name, address, and telephone number.

MicroPACE, Inc., will pay $10 per review published at the time of publication, or, upon request, credit $10 to your subscription. Be timely, be detailed, but be CONCISE!

Mail all subscriptions, requests, and reviews to:
MIDNITE SOFTWARE GAZETTE
P.O. BOX 1747
CHAMPAIGN, IL 61820
Reviews may be uploaded to Starship BBS at (217) 356-8056, or to Compuserve: 76703,4033; Q-Link: MIDNITE; Delphi: MIDNIT



The First Independent U.S. Magazine for users of Commodore brand computers.

Expand Past Maximum Capacity!

The Transactor

The Tech/News Journal For Commodore Computers

At better book stores everywhere! Or 6 issues delivered to your door for just $15.00
That's 29% off the newstand price! (Overseas $21 U.S. Air Mail $40 U.S.)
The Transactor. 500 Steeles Ave. Milton, Ontario. L9T 3P7. 416 878-8438
Also check out The Transactor Disk; every program from each issue, in order as they appear
and The Complete Commodore Inner Space Anthology; over 2.5 million characters of reference information exclusively.
Included are memory maps for BASIC, COMAL, CBM disk drives, BBS numbers, machine language charts, Kernel routine summaries, printer commands, recording formats, I/O port maps, port pinouts, audio control tables, video reference charts, keyword values, commands for common software packages, MLM and assembler commands, and there's more!

To us, expansion knows no limits!