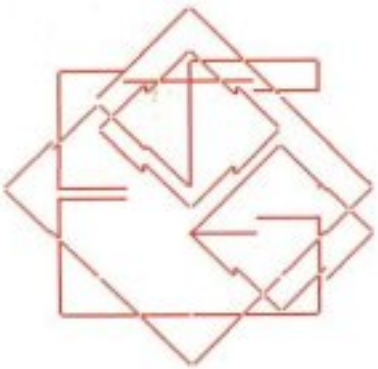
COMAL TODAY 16

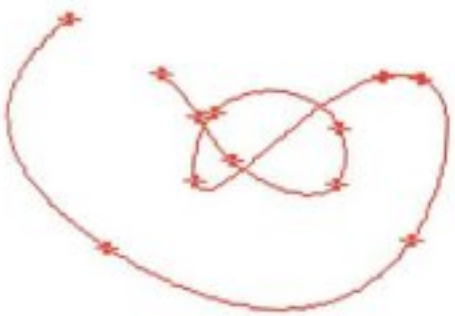
phew! see page 60

OVER ONE DOZEN TYPE IN PROGRAMS

COMAL Today
6041 Monona Dr.
Madison, WI 53716



WHAT IS IT? see page 40



WHAT IS IT? see page 66

Bulk Rate
U.S.Postage
Paid
Madison, WI
Permit 2981

If your label says
Last Issue: 16
You must renew now.
Use order form inside

COMAL Today Issue 16 March 26, 1987

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
Phyrne Bacon
Richard Bain
Jack Baldrige
Thomas Bishop
Reed Brown
Captain COMAL
Doug Drake
Mark Finley
Steve Furbish
Gerald Hobart
Bob Hoerter
Bill Inhelder
Russ Jensen
Sol Katz
Dick Klingens
Si LaBar
J William Leary

CONTRIBUTORS
Len Lindsay
Susan Long
Sam McLauchlan
Bill Nissley
Charl Phillips
Rahl
Joel E Rea
Val Rosad
Bruce Rosen
Robert Ross
Robert Shingledecker
Mark Sims
David Stidolph
Colin Thompson
Herbert Wollman
Dr Zarkov

From the Editor's Disk

by Len Lindsay

We are continuing to use small icon pictures at the top of articles to identify which version of COMAL it applies to. The **disk** means 0.14, **cartridge** means 2.0, **I** means UniComal IBM PC COMAL, and the **chip** means 2.0 with Super Chip.

By the time you read this, we should have two new COMAL implementations for distribution here in the United States and Canada... CP/M COMAL 2.0 and a new implementation of COMAL 2.0 for IBM PC and compatibles. Both should sell for under $100!

The CP/M COMAL 2.0 is rumored to include a true compiler that can compile a COMAL program into a stand alone program that can be run directly from CP/M without the need of the COMAL system. Hopefully it will also run on the C128 in CP/M mode.

The original COMAL from IBM Denmark has now been discontinued and is being distributed directly by UniComal Aps. Unfortunately, its price also increased another $100 or so. At that price, we will not be stocking it here, but can special order it for you (these orders must be prepaid, estimated cost of $500 - $600). It now comes with a 600 page manual **in English**.

Meanwhile, we now should have an inexpensive alternative COMAL 2.0 for the IBM PC from Mytech. Then later this year we look forward to MacIntosh COMAL, hopefully followed by Amiga COMAL, all from Mytech.

All COMAL implementations thus far have come from Scandinavia. However Apple computers are not popular in Europe. So, none of those companies plan to develop COMAL for the Apple. David Stidolph decided to do something about the lack of COMAL for the Apple II computers. He is writing the implementation himself! He has been sharing his expertise in COMAL since our second issue of *COMAL Today* where he gave us the start of a memory map for COMAL. He set up a one man company and is only working here half time so he can spend time on development work. He provides more information about his project on page 77. Even if you don't have an Apple computer, perhaps your friends or school does! We need Apple COMAL. **He would like your support ... and suggestions!**

Thanks for using our VOTE card from last issue. See page 5 for a summary of all the cards we got back. We are interested in what you need. This issue shows how we can adapt. Most people favored program listings in **bold** type style. So, we are using it when possible.

I'll be at the Commodore Show in San Francisco February 20, 21, and 22. Any COMALites in the area are welcome to come to the COMAL session at the show.

Do you have your **Index** to *COMAL Today* yet? We still have a good supply of the first printing ... with the **silver** cover. Perhaps you hesitate getting it since you don't have those back issues of *COMAL Today*. **You don't know what you are missing.** Those back issues include a treasure trove of COMAL information. This month, if you get the **Index**, we will throw in a copy of every back issue (#5-#12) that you need for only 50 cents each. If you have a USA address that UPS delivers to we won't even charge you the usual $1 each shipping! (First Class and Canada still have the $1 per issue shipping fee).

The new Commodore RAM expander for the C128 and C64 apparently can be accessed from both COMAL 0.14 and 2.0. Preliminary access routines are in our program libraries on QLink, and next issue we should have final versions for you. Maybe even a 2.0 package! If you can't wait till then, download the procedures from QLink and read the associated messages.

Finally, note that our order processing system is a huge COMAL program. As it is now, we have no way of separating backorders from those in stock, so all are processed the same (prepaid). ■

COMALites Unite

by Richard Bain

COMAL Today is now in its fourth year of publication. Over 1000 pages have been published. We have come a long way - from the cramped style produced on a daisy wheel printer to the proportional spaced, varied fonts of our laser jet printer. We have gone through several word processors; now we use *Word Perfect* on an IBM compatible and can put the whole newsletter on a subdirectory of our hard disk. But our main advantage is the extreme power and user friendliness of the COMAL language, and the dedication and support of our loyal readers who continually send in programs, letters, and questions for the benefit of all.

But where does COMAL stand in the world of computer languages? For years it has been popular in Europe; it is the official language taught in schools in several European countries. In the USA, COMAL has had more of a struggle. It has been restricted mainly to Commodore computers. Teachers at schools with Commodore computers have had to struggle for the right to teach COMAL instead of more traditional languages such as BASIC and Logo. Schools which only have Apple computers do not have the option to choose COMAL. Currently David Stidolph is trying to do something about this situation. He is spending every free moment he has working on a version of COMAL for the Apple computer. He won't rest until he sees all the schools with Apple computers start to teach COMAL.

Yes, the future of COMAL looks promising. COMAL 0.14 is an ideal language for the person who gets a first computer. It is inexpensive, yet powerful. Within an hour of setting up the computer, someone could be doing turtle graphics in COMAL.

Soon the programmer progresses to the intermediate level. It is time to move up to the COMAL cartridge. The advantages are obvious as soon as you turn on the computer. It powers up in COMAL; no more waiting for it to load. You get more free memory too. You will see the next big advantage as soon as you type in your first program. I don't mean to offend a perfect typist, but when you make that first typing mistake, COMAL will tell you. It gives explicit error messages telling you what you typed, and what is likely to be the correct replacement. If you forget whether you need a colon, or semicolon for a certain line, don't waste time looking it up. Make a guess and let COMAL tell you how to correct it.

As you become more familiar COMAL 2.0, you will be amazed at the new features. There are more graphics, sprites, and sound routines built in. There are even commands to support an alternate screen font, or to read a light pen, joystick, or paddle. If you don't care for fun and games, try using the error handling routines to prevent a program from crashing if the user puts the wrong disk into the disk drive. If this isn't enough, you can **LINK** machine language to your program, call its routines by name, and pass COMAL variables to it as parameters.

As you can see, COMAL can take you from your first program to advanced applications. An important factor in this progression is the compatibility among the different versions of COMAL. Many programs will run without change from COMAL 0.14, COMAL 2.0, and IBM PC COMAL. For most other programs, the conversions are minor.

This is a special time for those interested in COMAL at the system level. You may now have a direct influence on the development of Apple COMAL. David Stidolph needs help in making decisions on how to implement non-kernal enhancements. You can give him your *wish list* of COMAL enhancements, or simply watch as he describes the progression of Apple COMAL and the inner workings of the parser, scanner and run time systems. See page 77. ■

Questions and Answers

SUPER CHIP

Today my Super Chip and shareware disks arrived. Great stuff. All I need now is to learn how to use them all. A couple of things I want to ask you about:

1) For some reason **Rabbit** will not work. I suppose it is some incompatibility with European and USA Commodore systems. I've discovered the same thing with the *sizzle* loader.

2) I get an error message when I enter:
USE colors
background(white)
If I do it from graphic mode I get no result.

3) The exchange rate by the time it gets to New Zealand makes the cost more than double. I'm prepared to pay because COMAL is so great. - Mark Sims, New Zealand

1) We have heard several reports that Sizzle and Super Chip's Rabbit do not work in Europe. It has something to do with the 50 Hz power supply. However, the version of Rabbit released in COMAL Today #12 does work in Europe.

2) There are three possible explanations.

a) You forgot to issue the command use graphics or use turtle before your background command.

b) You may be incorrectly using the background command. Background does not affect the text screen in COMAL 2.0. Use the textbackground command from the Turtle or Graphics package.

c) Super Chip initialization sets the textscreen in case you call autostart with the graphics screen active. The side effect is that the F1, F3 and F5 function keys lose their default definitions. If this seems to be a problem, type NEW before doing anything else and all should be well.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

ARE GRAPHICS ACTIVE?

Phyrne Bacon asks if there is an easy way to detect if the graphics screen is active. This test could make it possible to have a graphics screen from the immediate mode remain for a running program. It also could ensure that graphics commands won't be used before initializing the graphics screen. This is important because it is erased the first time it is accessed from a running program. Calling graphics procedures prior to initializing the graphics screen causes the program to crash.

In COMAL 0.14, this function tells if the graphics screen is active.

```
func graphics'on
  return peek(27252)
endfunc graphics'on
```

This function returns 0 if the graphics screen has not been initialized, 1 if it is initialized but the text screen is currently being displayed, or 2 if the graphics screen is initialized and showing.

In COMAL 2.0, we haven't found a simple peek to determine if the graphics screen is active, but this poke prevents the graphics screen from being erased by the USE GRAPHICS command:

```
poke $c298,0
```

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

USER FILES

Si LaBar of Bethlehem, PA has these questions about USR files:

1) What are user files?
2) How are they written and why are they needed?
3) How are they saved to and retrieved from disk?

more»

4) Have they any really practical applications other than as mentioned in the next paragraph as part of a disk directory? Parts of the directories of all the COMAL disks, as well as other commercial disks I own, show USR files which seem to have no purpose other than setting off a block of text from other sections of the directory.
5) Are the USER1 and USER2 commands mentioned in the 1541 disk drive manual used to write user files?

Here are a few simple answers to your questions:

1) A USR file is a type of sequential file.
2) To open a USR file for writing:
open file 5,"user'name,u,w",write
To open a USR file for reading:
open file 5,"user'name,u,r",read
Once the USR file is opened, the READ, WRITE, INPUT, and PRINT file commands work the same as for SEQ files. You never need USR files, but sometimes it is convenient to have a different file type to distinguish files for different uses.
3) This question was answered in part two.
4) The practical applications of USR files can be as varied as a programmer's imagination and talents. The most common application that we are aware of is to help organize a disk directory the way we do our Today Disks.
5) The USER1 and USER2 are advanced disk drive control commands. They are not related to USR files.

NOW FOR OUR QUESTIONS

1) What are your favorite items in issue #15?

#1 - Procedures and Functions - Our Collection
#2 - TRON - The New Trace
#3 - QLink - The Message Base
#4 - Interrupt Package

2) What COMAL do you use?

C64 0.14
C64 2.0
SuperChip
IBM PC

3) How many program listings do you want?

More
Same
Less

4) How do you like the following listing styles?

Outlined Style
Good
OK
Na

Bold Text Style
Good
OK
Na

Small Text Style
Good
OK
Na

Line Printer Style
Good
OK
Na

As a result of the listing style survey, we find that most people prefer a normal size bold style. Many comment that it is easier to read than the small type. This issue we are trying to do most of the listings in **bold**, *avoiding the smaller print size. Also, note that the outline style listing was the most controversial. Many readers liked it for its emphasized structure, while others found the arrows and lines distracting.* ■

COMAL Clinic

EMPTY FILE?

Bruce Rosen of Southern Australia provides us with this note: When using sequential ASCII files, do not open an empty file for writing and then close it with the intention of appending to it later. If you do so, you will find these four bytes placed at the start of the file: CHR$(13), CHR$(0), CHR$(2), CHR$(13). When the correct data is later placed into the file, the four bytes will still be there. This causes problems when trying to read the data later. *[Commodores drive is the culprit, not COMAL. IBM does not have this problem.]*

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

NO PEEK, POKE, SYS

Dick Klingens tells his second level programming students to write programs without the use of **PEEK**, **POKE** and **SYS**; sometimes they don't want to listen. His package *pkg.no'pps* helps him. Give the command:

LINK "pkg.no'pps"

and the statements and commands containing, PEEK, POKE or SYS become *illegal operations.*

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

DISCARD KEEP FONT

Dick Klingens provides this procedure to discard a **KEEPFONT** from COMAL 2.0:

```
PROC nofont
  POKE $c2bb, PEEK($c2bb) BITAND 127
ENDPROC nofont
```

To use it from the immediate mode only, enter:

```
nofont
discard
```

SAVE/REPLACE @ IN 2.0

It could just be that COMAL 2.0 automatically avoids the save with replace "bug" that comes with all Commodore disk drives. If you do not specify the drive number, COMAL 2.0 will automatically "tack on" the default drive for you! This avoids problems with the drive getting mixed up on drive number. Next, COMAL 2.0 waits to receive the status on the disk access before doing anything else! This avoids problems the disk has in *getting ahead of itself*. So, as Bob Hoerter points out, the @ save with replace works fine with COMAL 2.0 (we think!).

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

TRY UNIT IN 2.0

COMAL 2.0 has a very nice feature in its use of the **UNIT** command. **UNIT** is used to set the default drive. For instance, if I have two drives, device 9 drive 0 is referred to as "2:" by COMAL 2.0. To access a file on that drive the filename can be prefixed with a "2:", such as "2:name". However, if you want to work exclusively with the device 9 drive, you can tell COMAL to use it as the default like this:

UNIT "2:"

Now, if you do not specify a drive number, it will default to drive "2:". If you have a Commodore datasette, you also can access it from COMAL 2.0. It is referred to as unit "cs:". You can use **UNIT** to set the default "drive" to be the cassette port:

UNIT "cs:"

For more information about **UNIT** see the *COMAL Handbook* page 305.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

more»

DELETE FILES IN 0.14

In COMAL 0.14 you **must** specify a drive number when deleting files from disk. This feature is included as a precaution in case you are using two drives. Thus the following line is ignored:

```
delete "testfile" // incorrect in 0.14
```

The following is correct:

```
delete "0:testfile" // correct in 0.14
```

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

A SUBSTITUTE FOR PASS

Execution of the statement:

PASS <*command*>

interrupts program execution in COMAL 2.0 until the DOS operation finishes. It is the same when **PASS** is given in direct mode.

Dick Klingens developed the following procedure for impatient programmers, because continuing work is possible while the command being passed to the DOS is executing.

```
PROC command(x$) CLOSED
 name$:="u8:"+x$+"/s15/d-"
 OPEN FILE 99,name$,WRITE
 POKE 152,PEEK(152)-1 // closes file 99
 // don't use CLOSE FILE 99!
ENDPROC
//
// example:
command("v0") // validate drive 0
```

The procedure is similar to the **PASS** command in COMAL 0.14 which does not pause the computer during the disk activity.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

CALL HIDDEN RAM

When calling a subroutine under the COMAL cartridge from a package, Robert Ross suggests using $66 instead of the normal $46 (DEFPAG) for the page definition. $46 does not work on the grey cartridge. He reminds us that calling $a000 to $bfff may send you to BASIC. Limit calls to the RAM from $8000 to $9fff. See *COMAL 2.0 Packages*, page 22, for more details.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

CART COLOR

The following function allows a running program to find out which COMAL cartridge is installed:

```
FUNC cart'color$ CLOSED
  USE system
  old:=PEEK($c840)
  SETPAGE($81)
  cart:=peek($a34d)
  setpage(old)
  IF cart=40 THEN
    RETURN "black"
  ELSE
    RETURN "grey"
  ENDIF
ENDFUNC cart'color$
```

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

PRINTING SPACES

In COMAL 0.14, multiple spaces can be inserted between printed words by using the semicolon (one space for each semi-colon). Here is an example:

```
PRINT "a";"b"
PRINT ;;;;"a";;;;"b"
```

The output looks like this:

```
a b
    a    b ■
```

Letters

FROM ONE TO ANOTHER

In September 1984 I introduced COMAL to one class of Seniors for about 27 weeks, and then finished the year with an intro to Pascal. I couldn't believe how fast they *"galloped"* through that material in 8 weeks time! All six classes this year are taking COMAL, using the COMAL 2.0 cartridge. We do not offer BASIC anymore. Next year we will offer Pascal with COMAL as a prerequisite. - J William Leary, Pocahontas County High School, Hemlock Brae, Rt 1, Box 137B, Dunmore, WV 24934

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

PROGRAM LISTINGS

Dear Editors- Your decision to take a vote on the issue of *to list or not to list* is most commendable. Since I am mainly a computer hobbyist, I must plan my expenditures carefully and try to get the most for my money. Much of what I do with COMAL is done by following examples that I find in the listings that you print, fitting things together to suit my needs.

At present I am considering subscribing to a telecomputing service and adding a 1571 disk drive to my C128. I don't want to have to give up my *COMAL Today* newsletter just to finance these other expansions, but without listings, the newsletter will lose much of its value to me. Adding a disk subscription is desirable, but that would require postponing any system expansion.

I enjoy COMAL, but it is only one portion of my computing ambitions. Presently I can do what I like with COMAL and still expand my system. Occasionally I buy a COMAL disk if it contains something that I want (such as the Database manager program). The Super Chip sounds like a future purchase also.

Getting back to the subject, I would like to see listings in *COMAL Today* increased or at least maintained at their current level. It will allow those of us on a budget not to be forced into deciding that COMAL has become too expensive. Thanks for allowing me to sound off. - Steve Furbish, Eliot, ME

Thanks for your comments. We anticipated that half our readers would want more listings, the other half wanting less. It turns out that over half (55%) of our readers want the same amount of listings. The other readers are evenly divided (24% want more, 21% want less). So, it looks like our best path now will be to maintain listings as we have in the past. So, don't worry about losing the program listings.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

COMAL IN GRADES 8 & 9

Dear Mr. Lindsay- This year we have started using COMAL 0.14 with our grade 8 and grade 9 computer literacy classes once a week. So far we are extremely pleased, especially with the graphics commands available. We have 18 Commodore 64's and a C128 in our lab. So far as I can tell, we have no further use for Logo. Even our program with Structured Basic will be more closely examined after I get more experience working with COMAL. - Sam A McLauchlan, Emmanuel Christian School, Dorval, Quebec

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

WHAT I LIKE ABOUT COMAL

I recently had a need to put a series of names and addresses into a two-column format. Since I don't have a word processor which will do that, I decided to take a look at Jim Ventola's *Double Column File Printer* from *COMAL Today #12*. I had previously played around with *User Disk #9*, which was the source for his program, so I had some idea of how it works.

more»

The difficulty was that I wanted to produce a disk file which I could load into a word processor for further massaging, rather than output hardcopy. The solution I came up with is a classic illustration of everything that's good about COMAL.

What I needed to do was to re-direct the output to a disk file. I examined the program listing and found the procedure **file'to'print**. This was easy because COMAL code is so readable. Then I found the line which set up the parameters for the device "lp:". This line reads:

open file 255,"",unit 4,7,write

That line sets up the printer with the upper/lower case character set. I tried changing this line to the following:

open file 255,name$+"-d",unit 8,write

Now the device "lp:" is the specified file on the disk drive! Running the program and selecting the print function produced the disk file that I needed.

My principal point is not what I did, but how easy it was. It took me more time to set up a test file to see if this idea worked, than it did to interpret the code and alter it. And I only had to change one line! I guess that's what I like about COMAL. - Doug Drake, Wisconsin

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

FAST PROGRAM ENTRY

Here are a couple of tricks I use to make life easier:

1) When writing a program, I use very short, easy to type names for procedures, functions and key variables. When the program is finished, I use the **CHANGE** command of COMAL 2.0 to change these to longer, more descriptive names for better documentation. You really can have it both ways.

2) When working on a long program the **hideaway** and **reveal** procedures in the Super Chip are handy. Let's say you're working on a procedure starting on line 3000. Issue the commands **use system2** and **hideaway(2999)** from the immediate mode. Then you can type **LIST** and not have to wait for the entire program to scroll by, nor do you have to type the procedure name every time you **LIST**. Just type **REVEAL** to get all your lines back again. - Gerald Hobart, Bellevue, NE

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

APPLE COMAL NEEDED

Dear COMAL Friends- I've really enjoyed using COMAL during the past year. I can't imagine ever going back to BASIC! I'm a graduate student in Educational Technology at Lehigh University. Last spring I gave a talk and demonstration on COMAL to other graduate students (many of whom had never used a Commodore computer, much less heard of COMAL - our lab uses only Apples and IBM's). As impressed as people were with its capabilities, I don't think too many American educators are going to use COMAL in schools until:

a) it becomes available for Apple IIe's (without CP/M)
b) the manual for the IBM version is published in English
c) courses are offered in COMAL at the university or community college level
d) it is available locally instead of only through your user group

I eagerly look forward to each new issue of *COMAL Today*. Keep up the good work. - Val Rosado, Bethlehem, PA

more»

Letters - continued

Thank you for the nice letter. We agree with most of your points - and progress is indeed being made.

(a) Apple COMAL is in the works. See page 77 for the project discussion.

(b) The UniComal IBM PC COMAL now comes with an English manual, but it's price has gone up (about $500 - $600). However, a new implementation of COMAL 2.0 for the IBM PC and MS-DOS should be available by the time you read this at a very reasonable price - less than $100. It is from Mytech and comes with an English manual. Mytech is also developing COMAL 2.0 for the MacIntosh, and possibly the Amiga. Plus, a new COMAL 2.0 is just being released for CP/M computers (which should include the C128 in CP/M mode). It too should be available for under $100.

(c) Courses in COMAL are now starting to appear at the college level. Also, Future Technical Institute is offering COMAL classes, as announced in COMAL Today #13, page 57.

(d) C64 COMAL is available from many sources including:

* *COMAL Users Group, U.S.A., Limited*
* *local Commodore User Groups*
* *AHOY monthly disks*
* *Transactor Disk #8*
* *Many TPUG disks*
* *LoadStar disk*
* *download from national networks including: QLink, People Link, Genie, Delphi* ■

Bug Fixes

BASIC'2'COMAL

Sol Katz sends these corrections to his *basic'2'comal-p1* program on *COMAL Today #13*. The corrected version is on *Today Disk #16*

In **PROC decode**, change the first **ELSE** to
elif ascii <> 32 then
In **PROC write'goto'or'proc**
change **gt'mat:=0** to
if ascii <> 44 then gt'mat:=0
change **ex'mat:=0** to
if ascii <> 44 then ex'mat:=0

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

SPEEDSCRIPT / SUPERCHIP

Doug Drake noted that the COMAL 2.0 Speedscript conversion programs on *Today Disk #9* (referred to on page 46 of *COMAL Today #9*) don't work on a c128 with Super Chip. They work fine on a c64. To use them from a c128 with Super Chip, issue these commands:

USE c128
Discard'c128

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

LOAD'ERRORS

The **load'errors** procedure listed in *COMAL Today #15*, page 48 is missing one line. Insert this line immediately after the procedure header:

dim ds$ of 2

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

PRINT FILE NUMBERS

Page 25 of *COMAL Today #15* said the 0.14 version of the print file number program is on *Today Disk #15*. We forgot to do that, but we will put the program on *Today Disk #16* ■

QLink Meeting Notes

Captain C: Some news... Talked to Mytech COMAL people today. Their MacIntosh COMAL had a major setback. They use a huge Hard Disk for the development. Of course, they always backed up their disk to streaming tapes. Well, as might be feared, their hard disk crashed - a very serious unrecoverable crash. Then the bad news ... they discovered that their backup program was not working properly. 20% of every file was lost. 80% of every file was not too useful, so it took them months just to get back to where they were before. That is why we didn't hear much from them on MacIntosh COMAL. Now, we hope to get a pre-Beta test version in a couple weeks. Mytech COMAL for the IBM PC is in much better shape - possibly a real live final version by Jan 87. We hope to have it here soon - and to be able to distribute it. The price will be right ... under $100 (preliminary from our discussion).
COMALite J: COMALite D: About your Apple COMAL project, how about a "SETKERNAL <+/->" statement which turns on/off warnings whenever a non-Kernal enhancement is entered, so people can be sure any program they write that needs to be completely portable IS completely portable?
COMALite D: How would you give this warning? A beep?? I would find it annoying.
COMALite J: Like an error message, but it would not reject the line. Like this:
SETKERNAL -
10 PRINT AT 12,5:"Howdy!"
SETKERNAL +
20 PRINT AT 5,10:"WOW!"
Warning! "AT" is a non-Kernal feature.
COMALite F: CD--are you going with line numbers or a full screen editor?
COMALite D: Both Just like C64 cartridge COMAL (and IBM for that matter).
COMALite F: CD--I will publicize it around Southern California. Send me details.
COMALite D: If you want to publish, I have a company: COMALites United, 1670 Simpson, #102, Madison, WI 53713. I will put a write up on it on QLink and in COMAL Today 16.

COMALite J: CC, on listings, I don't mind the style used, so long as it is MONOSPACED at 40 columns!! That makes it easier to visually check your typing!
COMALite B: Not really Joel, it is hard to match the indentations. The best idea is to get the disk and avoid typos.
Captain C: Joel, why 40 columns? Why not monospaced at 44 columns?
COMALite F: C64 prints 40 columns.
Captain C: But should we limit our listing to the limit imposed by the computer. The listing should be easy to read (and for some, easy to type in) The number of columns would only help if you wanted to match up to your screen exactly, and due to indentation differences, and other things, this is not really necessary, and makes for lots of extra typing. Also, we continue to NOT include line numbers most of the time, and no one has really complained about that. I assume all are happy with the AUTO command for program entry.
COMALite J: I forgot about that, sorry.
COMALite B: Don't be sorry, we welcome suggestions.
Captain C: COMAL is never having to be sorry.
COMALite J: :)
Captain C: Other news. Our COMAL 0.14 & 2.0 library here on QLink is now available for you to upload programs into - or download programs from. I have a special disk so I can view files before they are officially visible. If you upload anything, send EMail to me about it. I will get QLink to make them visible. The complete 7 file COMAL 0.14 system (from COMAL Today 15) is now uploaded onto the 0.14 library here and on other networks (Genie, PLink, Delphi).
Rah1: Has anyone done a program to list all variables and line numbers that use them? It would be handy combined with TRON.
Captain C: Rah - yes, we now have a full line listed TRON. The cross reference program was never listed in the newsletter, just added on a Today Disk. One of the benefits of getting the disk! The Cross Reference program has Dutch variable names I believe.

more»

Rah1: Do you know what disk it is on?
COMALite D: Today Disk #8, 2.0 side. It is called "cross'reference"
COMALite B: Also we got an updated TRON from a user who combined it with META to list the entire line.
COMALite D: With the new Meta/Tron, each line is listed to the screen as it is executed.
COMALite J: How about a TRACEIT? like this:
TRACEIT xcor
would show the current value of variable XCOR whenever it changes, plus the line number that changed it:
AT 0370: XCOR becomes 7.2
Or, you can trace a FUNCtion:
TRACEIT getcolor
AT 0650: GETCOLOR returns 7
COMALite B: I Think that could be done CJ, but it would be more work than we could do.
COMALite J: Or a PROCedure:
TRACEIT forward
AT 0275: FORWARD called
Or multiple items:
TRACEIT xcor,ycor,currow,curcol
Or, ALL PROCedures:
TRACEIT PROC
Or, ALL VARIABLES:
TRACEIT DIM
Or, EVERYTHING!
TRACEIT FUNC,PROC,DIM
Turn tracing of anything or everything off:
TRACEIT- xcor
TRACEIT- FUNC
TRACEIT- //everything
COMALite D: I prefer TRACE+ and TRACE- with an optional filename following to list lines to.
COMALite B: In Apple COMAL, trace could be built-in, one meaning .. within a prog and another when it finishes.
COMALite J: I'm not talking APPLE COMAL, but an addition to COMAL 2.0, like the TRON in CT#15!
COMALite D: CJ - that would be VERY difficult to do. If not impossible.
Rah1: Can TRON list to a printer so it doesn't mess up screens?
COMALite B: It could by using print file or select "lp:"
COMALite D: Yes, you could put a OPEN FILE command and PRINT FILE and CLOSE FILE in the procedure.
COMALite B: The idea of TRON is to discover where it was called from... If it also analyzed the line, it could give that info. Want to do the necessary recreation?
Rah1: CC - Thanks for the booklets for the Boston MARCA show. We had about 60 at the seminar and to a lot of people at the info table.
COMALite F: Did I understand correctly that the CP/M COMAL has a compiler?
Captain C: CF- Yes, Borge told us that the CP/M COMAL just released in Denmark had the capability to create a .COM file
COMALite F: Will it run on any CP/M machine? RAM limits? C128?
Captain C: CF - too many questions until we actually get a copy here. We only talked a few minutes overseas to Borge about it. Once we get it we should be able to supply answers to all these questions. Right now, its features are only hearsay from Borge.
Rah1: CC - Will it be distributed here?
Captain C: Rah - I hope that we will be its distributor here. We anticipate its list price at under $100. The list price of the new Mytech COMAL for IBM should also be under $100 when it is final - and that may be in Jan 87 That will be nice -- C64 2.0, IBM 2.0, and CP/M 2.0 all priced under $100. If the CP/M version works in the C128 it could be interesting... but it won't know about Graphics or Sprites or Sound.
COMALite D: But Borge said it has packages.
COMALite F: I take it Amiga COMAL stalled behind the crash of Mac COMAL, correct?
Captain C: CF - you got it. But, more news (Amiga type). Mytech is a Swedish company. They told us that the Amiga is selling very well in Sweden. And that the Atari ST is virtually not available. Therefore, they could do Amiga COMAL after Mac COMAL. They said that AMIGA computers sell for less in Sweden than here. Usually it is the opposite.

more»

COMALite J: I think the 1040 ST is a lot of fun to use! -- (Almost as much fun as the 1040 IRS!)
RobertS50: Do all versions of Comal all have the same run-time compiler? Is there any version that has a true compiler?
Captain C: Robert - the new CP/M COMAL just released in Denmark is said to have a true compiler ... into .COM files We hope to have a version here soon. It supposedly has been shipped to us. There are several different companies implementing COMAL on various computers. Each company will have its own method of implementation.
COMALite D: I am hoping to provide some-such system (compiler or run-time system) for my Apple COMAL.
Captain C: Unicomal's is similar to RC's I believe. But here in the USA, UniCOMAL is bout the only COMAL available now. But Mytech could change that soon, as could the new Danish company with CP/M COMAL.
COMALite F: Info mag had a great comment about a COMAL compiler.
Dr Zarkov: CF what vol. of info?
COMALite F: #12. They said that if COMAL had a compiler you couldn't force programmers to use anything else at ... ...gunpoint.
COMALite D: Any questions about Apple COMAL or suggestions? I have to go home soon.
RobertS50: Apple Comal ? Is there such a thing?
COMALite D: I am working on COMAL for the Apple IIe/c computer.
RobertS50: Is this for your Master Degree David?
COMALite D: I wish - I'm out of school for now.
RobertS50: We will have to call you Dr David.
COMALite D: I am planning on developing it to run under ProDos for the Apple IIe and Apple IIc. My current plan is for 32K of programming space for the user.
RobertS50: That should really give COMAL a boost!
COMALite D: One reason I want to do this with the support of the users, rather than a business, is that I want control of the source code to go on to other machines.

OTHER QLINK NOTES:

We hold a national COMAL meeting every first Sunday and every second Thursday of the month. The meeting starts at 10 pm Eastern time. It is held in our conference room. To get there do this:

Sign on to QLink.
Goto **Commodore Information Network (CIN)**
Choose **Magazine Rack**
Choose **COMAL Today**
Now, you can choose **Conference Room** or our **Messages Board** or our two program libraries. ■

COMAL Structures - IF

by Richard Bain 

The **IF** structure can be one of the easiest and most powerful structures to use in COMAL. It lets a program make a decision and act accordingly. The problem for a programmer using **IF** for the first time is that the structure has several optional parts. These options can be more confusing than helpful. Let's start with the shortest **IF** structure and work our way up to more complicated ones.

```
friend:=TRUE
foe:=FALSE
person:=friend
if person=friend then print "hi buddy"
```

The last line above contains an example of the **one-line IF** structure. It starts with the word **if**, followed by an «*expression*» (a test), the word **then**, and a command. The «*expression*», **person=friend**, must evaluate to either **TRUE** or **FALSE**. This is as *logical* as it seems. Since **person** was assigned the value of **friend**, the test will evaluate to **TRUE**. Had person been assigned the value of **foe**, the test would have evaluated to **FALSE**. After determining that the test evaluated to **TRUE**, the **IF** structure executes its command, **print "hi buddy"**. Had the test proven **FALSE**, nothing would be printed.

A **one-line IF** structure is very handy when there is one task for a given condition. When the task is more complicated, you can use a **multi-line IF** structure:

```
friend:=TRUE
foe:=FALSE
person:=foe
if person=friend then
  print "hi buddy"
  print "how are you today?"
endif
```

The **multi-line IF** structure starts like the **one-line IF** structure except it doesn't have a command on the line with the **IF**. You can put as many commands after this as you want. You can even include nested **IF** structures (one **IF** structure inside another). The **multi-line IF** structure ends with the **ENDIF** keyword. In this example, **person** has been set to **foe** so the **PRINT** statements will not be executed. If the **person** was a **friend**, the **PRINT** statements would be executed.

So far, the **IF** structure has only been used to call a section of code if some condition is **TRUE**, or skip that section if the condition is **FALSE**. The next example shows how the condition can be used to choose a section of code.

```
if person=friend then
  print "hi buddy"
  print "how are you today?"
else
  print "hi, let's be friends."
endif
```

In this example, you can still greet an old **friend**, but now you have the added option of making a new **friend**. If the **person** is your **friend**, the first two **PRINT** statements will be executed, but the last one will be skipped. If the **person** is not your **friend** yet, only the last **PRINT** statement will be executed.

Some **people** may not want just anyone as a **friend**. The **IF** structure allows for this:

```
if person=friend then
  print "hi buddy"
  print "how are you today?"
elif person=foe then
  // walk away
else
  print "hi, let's be friends."
endif
```

This example introduces the optional **ELIF** portion of the **IF** structure. The **person** is first

more»

checked as a **friend**. Only people who are not friends will be checked as foes. You still get to try to make new friends, but you no longer talk to your enemies.

Advanced notes:

You are not limited to one **ELIF** in an **IF** structure. You can have as many tests as needed before the **ELSE** statement. In fact, **ELSE** is always optional. You can have one or more **ELIF** clauses in an **IF** structure without an **ELSE**. If none of the tests evaluate to **TRUE** then nothing is done. Only one **ELSE** may appear in an **IF** structure.

So far, this example could have been replaced by the **CASE** structure. I won't recommend one structure over the other, it is a matter of preference:

```
CASE person OF
WHEN friend
  print "hi buddy"
  print "how are you today?"
WHEN foe
  // walk away
OTHERWISE
  print "hi, I want to be you friend"
ENDCASE
```

This final example of the **IF** structure cannot be converted properly to a **CASE** structure:

```
input "type a number: ":x
if x>0 then
  print "number is positive"
elif x=0
  print "number is zero"
else
  print "number is negative"
endif
```

Further reference:

Making Decisions, COMAL Today #10, page 58 ■

How To Type In Programs

Line numbers are required for **your** benefit in editing a program (but are irrelevant to a **running** 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
   IBM 2.0: Hit «*ctrl*»-«*break*»

COMAL 0.14 note: 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.

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'] [\ _

The «left arrow» key in the upper left corner of the C64 / C128 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 \. ■

Text Input Window

by David Stidolph 

Edit'box on *Today Disk #16* gives the COMAL 0.14 user a *text box* to type in, and continues to let the user type until a certain key (f1) is pressed. This idea could be used in different applications such as a card file or data base program which allows comments. The user may not go outside the defined box (it's like the protected INPUT field in COMAL 2.0, but with more control over the field). After the f1 key is pressed, the program moves the cursor down the screen and prints the text from the box.

```
dim txt$(5) of 20
dim v$ of 256
v$:="0123456789!#$%&'()+-@*^:;[]"
v$:=v$+"=,./<>?abcdefghijklmnopq"
v$:=v$+"rstuvwxyzABCDEFGHIJKLMNO"
v$:=v$+"PQRSTUVWXYZ """ // +"
print chr$(147),chr$(14),
print "Type any text you want into this"
print "box. Press f1 to exit EDIT mode."
edit'box(5,5,20,5,txt$,v$,chr$(133))
//
cursor(15,1)
for x:=1 to 5 do
 print txt$(x)
endfor x
//
proc edit'box(row,col,wide,lines,ref array$(
),valid$,finish$) closed // wrap line
 minrow:=row; maxrow:=row+lines-1
 mincol:=col; maxcol:=col+wide-1
 dim c$ of 1
 cursor(minrow,mincol)
 for x:=1 to lines do
  array$(x)(1:wide):=array$(x)
 endfor x
 repeat
  r:=row-minrow+1
  inkey(c$)
  case c$ of
  when chr$(19) // home
   row:=minrow
   col:=mincol
  when chr$(147) // clr/home
   for x:=1 to lines do
    array$(x)(1:wide):=""
    print'at(minrow+x-1,mincol,array$(x))
   endfor x
   row:=minrow; col:=mincol
   cursor(row,col)
  when chr$(20) // delete
   p:=curcol-mincol+1
   if p=1 or p=2 then
    array$(r)(1:wide):=array$(r)(2:wide)
   else
    array$(r)(1:wide):=array$(r)(1:p-2)+array$(r
    )(p:wide) // wrap line
   endif
   print'at(row,mincol,array$(r))
   col:-1
   cursor(row,col)
  when chr$(148) // inst
   if array$(r)(wide)=" " and col<maxcol then
    p:=col-mincol+1
    if p=1 then
     array$(r):=" "+array$(r)(1:wide-1)
    else
     array$(r):=array$(r)(1:p-1)+" "+array$(
     r)(p:wide-1) // wrap line
    endif
    print'at(row,mincol,array$(r))
    cursor(row,col)
   endif
  when chr$(17) // cursor down
   row:+1
  when chr$(145) // cursor up
   row:-1
  when chr$(29) // cursor right
   col:+1
  when chr$(157) // cursor left
   col:-1
  when chr$(13) // carriage return
   row:+1; col:=mincol
  when finish$
   return
  otherwise
   if c$ in valid$ then
    print c$,
    array$(r)(col-mincol+1):=c$
```

more»

Text Input Window - continued

```
    row:=currow; col:=curcol
   endif
  endcase
  if curcol<mincol or col<mincol then col=mincol
  if curcol>maxcol or col>maxcol then col=maxcol
  if currow<minrow or row<minrow then row:=
  minrow // wrap line
  if currow>maxrow or row>maxrow then row:=
  maxrow // wrap line
  cursor(row,col)
 until true=false
endproc edit'box
//
proc cursor(row,col)
 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
//
func currow
 return peek(214)+1
endfunc currow
//
func curcol
 return peek(211)+1
endfunc curcol
//
proc inkey(ref c$)
 c$:=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
//
proc print'at(r,c,t$)
 cursor(r,c)
 print t$,
endproc print'at ■
```

Submitting Articles

Would you like to share information, programs, or articles with other COMALites? COMAL 0.14 material is especially appreciated. Many COMALites have moved up to the cartridge, but there still are many 0.14 users. Send all submissions to:

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

If you submit a program, please send it on disk. A printed listing of the program is not necessary. If possible, also include a text file explaining the program. Put your name and address as remarks at the beginning of your programs. This helps us give proper credits if they are used. Most important: label the disk with your name, address and date.

Articles should be submitted as standard SEQ text files on disk. If possible, also include a printout of each file on the disk. Don't include any special formatting commands in your files (we have to delete them). We use special formatting with PaperClip on our Commodore computer and Word Perfect on our IBM computer for our LaserJet printer.

Don't worry if you aren't a professional writer. Articles sent to us go through extensive editing. We actually go through over 4,000 sheets of paper while preparing one 80 page newsletter! You don't have to follow a bunch of rules, either. We rework your submissions to fit our newsletter format.

Material submitted is not returned. However, if you send us a disk, we will send one of our User Group disks back to you in exchange. Just specify which one.

Submitted material may also be used for other disks including our **READ & RUN** series. ■

Screen Help Package

by Gerald Hobart 

Prior to becoming familiar with COMAL, I used several BASIC extension products including *Graphics BASIC* by HES and *Simons' BASIC* distributed by Commodore. While I find COMAL to be a much more satisfactory programming environment, I still miss some of the commands from those products. I wrote this **Screenhelp** package so I could enjoy those commands with COMAL 2.0. It is on *Today Disk #16*.

Screenhelp contains 13 commands for controlling output to the textscreen. It must be linked to the current program with the command:

LINK "pkg.screenhelp"

and the program must issue the command:

USE screenhelp

prior to using any commands from this package. (The commands may also be given from the immediate mode.) Since the package has not been *rommed* it will be saved along with any program to which it has been linked. The commands are described below.

Textwindow(<row1>,<col1>,<row2>,<col2>) defines the textscreen window which will apply to subsequent commands from this package. **Row1**, **col1** specifies the upper-left hand corner of the window. **Row2**, **col2** specifies the lower-right hand corner of the window. Rows are numbered 1-25 and columns are numbered 1-40. Until a **textwindow** command has been given, the default window is the entire screen (1,1,25,40). (Note: If a program linked to this package is saved, the textwindow in effect at that time will become the default window.)

Windowframe(<color>) draws a frame around the inside of the currently defined window in the color specified by **color**. If one uses this command it, is normally desirable to re-define the window slightly smaller afterwards; otherwise the frame itself will be affected by subsequent **Screenhelp** commands.

Change'frame toggles the type of frame produced by **windowframe**. A double frame is the default type. **Change'frame** will change it to a thin, single frame. This does not affect frames already drawn, only subsequent frames. If you want to design your own windowframes, you can poke the desired screen codes to the following locations:

$963b - Upper Left Corner
$963c - Upper Right Corner
$963d - Lower Left Corner
$963e - Lower Right Corner
$963f - Top/Bottom Lines
$9640 - Left/Right Sides

For example try poking either 127 or 160 to all these locations, then issue a **windowframe** command. *(Note: If a program linked to this package is saved, whatever frame type is in effect at that time becomes the default frame type. This includes custom frames.)*

Next come the scrolling commands: **scrollup**, **scrolldown**, **scrollleft**, and **scrollright**. These commands scroll the material within the presently defined window one row or column in the direction indicated. The row or column on the leading edge is lost and a blank row or column appears on the trailing edge.

The rolling commands, **rollup**, **rolldown**, **rollleft**, and **rollright** work exactly like their scrolling counterparts except that the material that is lost at the leading edge of the window re-appears at the trailing edge.

Reverse will *invert* the mode of all material within the current window. That is, all material printed in the normal mode will be switched to the reverse mode, and vice-versa. Thus, using this command a second time will bring the material back to its original state. Since the

more»

window can be defined to be a portion of a single row, this command is useful for making a moving-bar type of menu.

Textfill(<char>) fills the current window with the character whose screen display code is specified by **char**. (See Appendix B of the *Commodore 64 Programmer's Reference Guide* for a table of screen display codes.) For example, **textfill(32)** clears the current window.

Colorfill(<color>) fills the color-ram corresponding to the current window with the color specified. The effect of this is to change the color of all printed characters within the window to the selected color.

There is also a **version'screenhelp$** function which returns a string containing the version number, author and other information, and a **texthelp** procedure which prints out all the commands of this package.

I located the package at $9200-$989a in an attempt to minimize any conflicts with other packages that might be used with this one. *Demo/screenhelp* on *Today Disk #16* shows how to use these commands for some exciting effects. The following shows the simplicity of the commands:

```
USE screenhelp
PAGE
textwindow(5,5,20,35)
windowframe(1) // white
textwindow(6,6,19,34)
// the frame is no longer part of the window
PRINT AT 19,13: "up up and away"
FOR x:=1 TO 13 DO
  scrollup
  FOR pause:=1 TO 100 DO NULL
ENDFOR x ■
```

TRON Revisited

by Steve Furbish

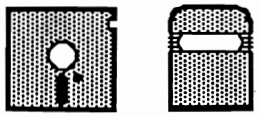
The *meta* package from *Today Disk #10* works well with the **tron** utility from *COMAL Today #15*. This gives the ability to **LIST** each COMAL statement, complete with line number, as it executes. To use the utilities together, you must **LINK** in the rommed version of *meta* before running the program, or **LINK** in the rammed version of *meta* to the program. Note, the four procedures related to the **tron** utility must also be included in the program.

Then to list lines (during the running program), insert the **tron** command before the lines you want to see. Note: you must put the **troff** command after the last line you want to see or the program may crash. (It repeats the last line of the program forever.) You may use **tron** and **troff** as many times as necessary within a program to limit the *trace* to sections of a program which are causing problems. Then you won't get a listing cluttered with sections of the program which have been fully debugged.

The four procedures for the **tron** utility and the linkable form of the *meta* package are on *Today Disk #16*. **Print'line'number** is the only procedure from the **tron** utility which needs to be changed from the listing in *COMAL Today #15*. It is listed below.

```
PROC print'line'number CLOSED
  USE meta
  pt:=PEEK($10)+256*PEEK($11)
  la:=PEEK(pt+4)+256*PEEK(pt+5)
  ln:=256*PEEK(la)+PEEK(la+1)-10000
  IF ln>0 THEN
    eval("LIST "+STR$(ln)+CHR$(13))
  ENDIF
ENDPROC print'line'number ■
```

Making Magic Squares

by Jack Baldrige 

I've been fascinated with magic squares since I was about 12 years old and read about them in a book of mathematical recreations. I recently found an article and a program dealing with them in an old issue of *Creative Computing*, so I wanted to share it with you.

Magic squares are square arrays of consecutive integers, starting with the number one, whose rows, columns and diagonals all add up to the same total. They may have an odd or even number of rows or columns. It is quite easy to make one with an odd number of rows, but to make one with an even number of rows (for example, 4x4) requires brute force methods.

My program on *Today Disk #16* constructs an odd-level magic square with a size between 3x3 and 29x29, using an algorithm similar to a simple geometric method. When I first thought of sending it to *COMAL Today*, I was puzzled about how to explain that method. I ended by writing a program to do it, *demo/magicsquare*. I enjoyed making the second program do more than the original adaption. This COMAL 0.14 program shows construction of a 5x5 magic square. The geometric method in the demo, can be used to make magic squares which are larger than the 3x3 square shown. But the method soon becomes pretty unwieldy. Try making a 7x7 or 9x9 square with it and you'll see what I mean.

The row, column and diagonal totals can be calculated without making the square. Each total equals (n^3+n)/2, where n is the number of elements in a row or column.

The following program constructs a magic square with a size from 3x3 to 29x29:

```
setup
fill'array
print'square
calc'sums
//
proc setup
  repeat
    print chr$(147)
    print "Magic Squares Larger than 9 by 9"
    print "Will be Output to Printer"
    print
    input "Size of Array? (3-29) ": n#
    print
    print "Working..."
  until n# mod 2 and n#>2 and n#<30
  dim square(n#,n#)
  for i#:=1 to n# do
    for j#:=1 to n# do square(i#,j#):=0
  endfor i#
endproc setup
//
proc fill'array
  i#:=n# div 2+1; j#:=1; square(i#,j#):=1
  for x:=2 to n#*n# do
    j#:-1; i#:+1
    if j#=0 then
      if i#=n#+1 then
        j#:+2; i#:=n#
      else
        j#:=n#
      endif
    elif i#=n#+1 then
      i#:=1
    else
      if square(i#,j#)<>0 then j#:+2; i#:-1
    endif
    square(i#,j#):=x
  endfor x
endproc fill'array
//
proc print'square
  if n#>9 then select output "lp:"
  print chr$(147)
  print "This is a";n#;"by";n#;"Magic Square"
  print
  for j#:=1 to n# do
```

more»

```
    print ">> ",
    for i#:=1 to n# do
      print using "####": square(i#,j#),
    endfor i#
    print
  endfor j#
endproc print'square
//
proc calc'sums
  print
  print "Sums:"
  for j#:=1 to n# do
    a#:=0; b#:=0
    for i#:=1 to n# do
     a#:+square(i#,j#)
     b#:+square(j#,i#)
    endfor i#
    print using "Row     ##=": j#,
    print using "#####": a#,
    print using "  Column  ##=": j#,
    print using "#####": b#
  endfor j#
  a#:=0; b#:=0
  for j#:=1 to n# do
    k#:=n#+1-j#
    a#:+square(j#,j#); b#:+square(k#,k#)
  endfor j#
  print using "Diagonal 1=#####": a#,
  print using "  Diagonal 2=#####": b#
  select output "ds:"
endproc calc'sums
```

9 X 9 Magic Square:

47	58	69	80	1	12	23	34	45
57	68	79	9	11	22	33	44	46
67	78	8	10	21	32	43	54	56
77	7	18	20	31	42	53	55	66
6	17	19	30	41	52	63	65	76
16	27	29	40	51	62	64	75	5
26	28	39	50	61	72	74	4	15
36	38	49	60	71	73	3	14	25
37	48	59	70	81	2	13	24	35 ■

Best Books

It's nice to see the *Graphics Primer* in the charts. It is a good beginners guide to using graphics and sprites with COMAL 0.14. We are surprised that the *Index to COMAL Today* isn't number one. We thought everyone would want a copy. If you didn't get a copy yet because you don't have the back issues, we now will make you an offer you can't refuse: we will give you any back issues you are missing (#5-#12) for 50 cents each when you order the Index. We won't even add extra shipping if you have a USA street address served by UPS (First Class and Canada still need $1 each for added shipping). This offer is for current subscribers only.

And the brand new book/disk set, *Graph Paper* should be ready in April. As a teacher, Garrett Hughes found this function graphing system worked so well that he wanted it to be available for others. Order it now, and get it hot off the press.

November 1986

#1 - COMAL From A to Z
by Gordon Shigley
#2 - Graphics Primer
by Mindy Skelton
#3 - COMAL Handbook
by Len Lindsay
#4 - Cartridge Graphics & Sound
by Captain COMAL's Friends
#5 - Cartridge Tutorial Binder
by Frank Bason & Leo Hojsholt

December 1986

#1 - Cartridge Tutorial Binder
by Frank Bason & Leo Hojsholt
#2 - Cartridge Graphics & Sound
by Captain COMAL's Friends
#3 - COMAL Today - The Index
by Kevin Quiggle
#4 - Introduction to Computer Programming with COMAL
by J William Leary
#5 - COMAL From A to Z
by Borge Christensen ■

CAI - Editor

by Reed Brown

This COMAL 2.0 computer aided instruction program functions on a stand alone system or in a networking situation. The program contains eight lessons on the major editing concepts and a twelve question quiz. The results are kept in a data base file. While the students always get the same twelve questions, the order in which they are asked varies. Even the replies for each question are randomly selected. Students are encouraged to re-run the program and re-take the quiz. Only the number of times they take the test and their highest score are saved. *Editor-cai* is on *Today Disk #16.*

The program was written to be *user proof.* Nothing the student can type should *bomb* the program. It will exclusively accept valid answers or use default values. The stop key has been disabled during the run of the program. Only the restore key still functions to break out of the program. The main problem that could occur would be when the program tries to access the data base. If the disk is not in the drive, an error will result. Also, if the student changes his/her student number, then he/she will appear twice in the data base.

The program listing has been disabled so that only a few comments can be listed. It is possible for the instructor to see the full program listing by typing:

SCAN
reed
LIST

To reverse this process, the command **hide (9200)** should be utilized.

Another option available to the instructor is control of the data base. When the black *quiz screen* is present, the student is instructed to hit «*return*» to go back to the lessons or «*space*» to go on to the quiz. Two other options are open to the instructor. Hitting the British pound key lists all the students in the data base to either the screen or a printer. The listing contains the student's number, name, I.D. code, high score, and times he/she took the quiz. The other option available during the *quiz screen* is hitting the negative key (-). This permits the removal of students from the data base in case of duplication.

This *CAI* program is designed to be used mid-way through an introduction to programming course. It allows students to be assigned numbers which will be requested by a data base. Only numbers from one to 250 will be allowed. The limit of 250 can easily be changed by the instructor. Change the statement

stud'count#:=250

to an appropriate value after using the **reed** command. The instructor could allow the students to randomly select their own numbers since the data base will not allow two students to use the same number.

If the *CAI* program is placed as the first program on the disk, the students will be able to auto-boot the program by using the shifted run key. The very first time the program is used on a disk, the program will automatically create the two files needed by the data base. A file called *students* is the random access file containing the data on all the students. A file called *pointers* is a short sequential file used to keep track of the random access file. Care needs to be taken to be sure the disk has sufficient room for the creation of the data base files.

Before the program is to be used, the instructor needs to briefly explain the purpose of the program, along with the information about the student numbers. Students could be told to start on lesson one and encouraged to take notes. Then the students could **LOAD** and **RUN** the program.

more»

Each screen of information is timed to give the student the proper amount of time to read the screen. This discourages students from rushing through the lessons. It should take the average student about 30 minutes to complete the eight lessons that the program contains. That leaves ample time in a normal class period to complete the quiz.

Note that each lesson automatically goes on to the next lesson. The final lesson defaults to the first lesson. In order to take the quiz, the students must go to the menu. After an opening quiz page, the program asks for the student's number. If that number is being used for the first time, the program then asks for the student's name and an I.D. code. The I.D. may be any two digits or letters. The purpose of the I.D. is to prevent students from altering another student's score.

At the end of the day the instructor should dump a listing of the data base information to a printer. It may be needed if students forget their secret I.D. code. All the instructor needs to do after the first day is to allow the students to re-use the program and encourage them to use the concepts that were covered. At some point the instructor needs to check the data base to see that all the students actually completed the quiz successfully.

[Editor's note: It's nice to see how COMAL is being used in the classroom. This program is especially good, because it should be easy for a teacher to change the lessons and quiz to any subject, even ones outside the computer field. The routines to keep track of the quiz scores can remain the same. Also, you don't need to be a student to learn from the lessons presented here. We found a few little used, easily forgotten, but helpful editing tips by running this program.] ■

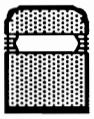
Easy Sprites

This short COMAL 0.14 program demonstrates how to make a bat fly on the graphics screen. It is on *Today Disk #16*.

```
dim image$ of 64
setgraphic 0 // graphic screen
define'image
show'sprite
move'sprite
//
proc define'image
  for x=1 to 64 do
    read temp
    image$(x):=chr$(temp)
  endfor x
  define 1,image$ // image #1
  data 0,0,0,0,0,0,0,32,64,0,246,240,3
  data 255,252,6,255,246,8,95,161,0,6
  data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  data 0,0,0,0,0,0,0,0,0
endproc define'image
//
proc show'sprite
  spritepos 1,50,50 // its location
  spritecolor 1,2 // its color red
  identify 1,1 // sprite 1=image 1
  spritesize 1,true,false // fat one
endproc show'sprite
//
proc move'sprite
  x=50; y=50; changes=2
  while true do // forever
    y:+changes
    spritepos 1,x,y
    changes=-changes
  endwhile // use stop key to end
endproc move'sprite
```

A sprite image is defined with a 64 character string pattern. Data statements provide the 64 needed characters. The **DEFINE** command sets up a sprite image. The **IDENTIFY** command assigns an image to one of the 8 sprites. We also set its color, position and size. Finally, the **move'sprite** procedure moves the sprite up and down. ■

Easy Instructions



by Captain COMAL

Wouldn't it be nice to write directions for your program with a word processor, and then have your program use the resulting SEQ text file?

Your instructions would be nicer, but then you would have to add special file reading routines into your program. And every time you copied the program to a new disk, you also would have to remember to copy the SEQ text file as well.

My solution utilizes the **Text** package released in *COMAL Today #11*. By storing your instructions as text inside the **Text** package memory (RAM memory hidden under the cartridge, unavailable to you as regular free memory), you do not lose any programming memory. Plus, when you save your program, the **Text** package memory will automatically be saved with the program.

Here are two short routines that you can use. One reads a SEQ text file and stores it into the **Text** package memory (each line in the text file must end with a carriage return, maximum of 39 characters per line). The other is a simple file reader, just like the *READ and RUN* system, only it handles the **Text** package memory text.

Now you can add 16K of instructions to your program, but use less than 1K of programming memory (for the file reader routine).

First, store the two procedures listed below to disk so that they are available to merge onto any future program. *You should have a disk that you use to store useful procedures on.* I will call this disk your *library* disk.

The first procedure will read the text file, line by line, and store it in the **Text** package memory. To store it on disk, type these lines:

```
NEW
AUTO

PROC merge'text(filename$) CLOSED
  USE text
  DIM textin$ OF 39
  OPEN FILE 22,filename$,READ
  rewrite
  PRINT "reading file into text buffer"
  WHILE NOT EOF(22) DO
    INPUT FILE 22: textin$
    writeln(textin$)
  ENDWHILE
  CLOSE FILE 22
  PRINT "done. you can save the program now."
ENDPROC merge'text
```

Hit the «*stop*» key to stop auto line numbers. Put your *library* disk in the drive then type:

```
LIST "proc.merge'text"
```

The **Text** package file reader displays the instructions and then restores the text screen to its original state. To store it on disk do this:

```
NEW
AUTO

PROC information CLOSED
  USE system
  USE text
  DIM textout$ OF 39, screen$ OF 1505
  getscreen(screen$)
  reset
  WHILE (NOT eot) AND (KEY$<>"^") DO
    IF PEEK(653) THEN
      readln(textout$)
      PRINT textout$
    ELSE
      PRINT "press shift key to read or ^ key
      to end"+CHR$(145) //wrap line
      PRINT SPC$(39)+CHR$(145)
    ENDIF
  ENDWHILE
  INPUT AT 0,0,0:"hit <return> to continue: ":
  textout$ //ignore it wrap line
  setscreen(screen$)
ENDPROC information
```

more»

Hit the «*stop*» key to stop auto line numbers
Put your *library* disk in the drive then type:

LIST "proc.information"

Now you are ready to try it out. Follow these steps to use the system in your programs:

1) Use a word processor to create an SEQ text file of your instructions (39 characters per line limit, carriage return ending each line).

2) From COMAL, load your program that needs the instructions. Then put a disk that has the **text** package file on it into the drive and type:

LINK "pkg.text"

3) Put your *library* disk in the drive and type:

MERGE "proc.merge'text"

4) Before you can use the **merge'text** procedure, you must **RUN** or **SCAN** the program. Type:

SCAN

5) Next, fill the **text** package memory with the text you created with the word processor. Insert the disk with your textfile on it then type:

merge'text("textfilename")

6) We no longer need the **merge'text** procedure in our program, so let's delete it. Type:

DEL merge'text

7) Now merge in the **information** procedure (put the *library* disk back into the drive):

MERGE "proc.information"

8) Add a line to the start of your program to call the newly added **information** procedure:

RENUM
5 information

9) Save your program (put your program disk back into the drive). The **Text** package and the instructions will automatically be saved with it:

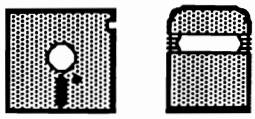
SAVE "programname"

10) Curious? You can see the instructions from direct mode. Just type:

SCAN
information

Demo/infomaker is on *Today Disk #16*, along with the two procedures and the **Text** package. If you don't have a SEQ text file to play with, you can use one of the two procedure files. ■

Index Disk Reader

by Len Lindsay 

We were fortunate that when Kevin Quiggle put together the **INDEX** to *COMAL Today*, he did it in a manner that resulted in a data file for each major index (title, author, and subject). Those data files are on the *INDEX disk*, along with the programs he used to create them. What was missing on the original *INDEX disk*, was a COMAL program to read the data files. The following program has now been added to the *INDEX disk*, and is on *Today Disk 16* as well. It will read an INDEX data file, and format it nicely for you, combining references to the same item. The program allows you to specify what issues of *COMAL Today* you have, and only references those issues in its printout. It also shows how to handle the "@" delimiter character that Kevin uses in the files.

If you have an original *INDEX disk* that does not have this program on it, we will update your disk for no charge if you return it to us. See the new *INDEX disk* directory on page 78.

```
//  delete "0:see'index(0.14)
//   by len lindsay
//    save "0:see'index(0.14)
dim text$ of 99, filename$ of 18
dim reply$ of 1, word$ of 80
dim index$ of 80, pagenum$ of 3
dim issue$ of 3, lastword$ of 80
print chr$(147)+chr$(14)+chr$(9)
textcolors(11,0,1)
get'filename(filename$)
get'issues(first'issue,last'issue)
open file 2,filename$,read
show'index(first'issue,last'issue)
close
//
proc show'index(starting,ending)
 print chr$(147)
 print "comal today issues";starting;"-";
 print ending;"index"
 print
 lastword$:="" // init
 while not eof(2) do
  input file 2: text$
  loc:="@" in text$
  if loc then
   word$:=text$(1:loc-1)
   if word$<>lastword$ then word'shown=false
   index$:=text$(loc+1:len(text$))
   repeat
    loc:="-" in index$
    issue$:=index$(1:loc-1)
    comma:="," in index$
    if comma then
     pagenum$:=index$(loc+1:comma-1)
     index$:=index$(comma+1:len(index$))
    else
     pagenum$:=index$(loc+1:len(index$))
     index$:=""
    endif
    if value(issue$)>=starting and value(issue$
    )<=ending then // wrap line
     if not word'shown then
      textcolors(11,0,1)
      print word$
      word'shown:=true
     endif
     textcolors(11,0,12)
     print tab(3),"comal today #"+issue$+",";
     print "page";pagenum$
    endif
   until index$=""
  else
   print text$
  endif
  lastword$:=word$
 endwhile
endproc show'index
//
proc get'filename(ref name$)
 print "a - authors index"
 print "k - keywords index"
 print "t - titles index"
 print
 repeat
  input "your choice (a / k / t) : ": reply$
 until reply$>"" and reply$ in "aAkKtT"
 case reply$ of
```

more»

```
 when "a","A"
  name$:="authors"
 when "k","K"
  name$:="keywords"
 when "t","T"
  name$:="titles"
 otherwise
  stop  // error
 endcase
endproc get'filename
//
proc get'issues(ref first,ref last)
 print
 repeat
  print "what is your first issue of comal today"
  input "(1-12) ? ": first
 until first>=1 and first<=12
 print
 repeat
  print "what is your last issue of comal today"
  print "(",first,"-12)";
  input "? ": last
 until last>=first and last<=12
endproc get'issues
//
proc textcolors(bor,ba,text)
 if bor>=0 then border bor
 if ba>=0 then background ba
 if text>=0 then pencolor text
endproc textcolors
//
func value(s$) 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
 endif
endfunc value ■
```

Dummy Files

by Jack Baldrige

You can organize the contents of a disk in an attractive manner by using dummy files. One of the best examples I've found is the USR files on *Today Disks*. These files can be made without too much trouble using a disk editor or the *make'dummy'files* program from *Today Disk #16.*

The requirements are pretty simple: I point all dummy files at block address of track 18, sector 18, as is done on the *Today Disks*. The file used is generally a USR file; that way, if you want to delete all the dummy files on a disk, you may do it with the command **pass"s0:*=usr"**. In order to avoid a directory error if the 1541 takes a careful look at the disk, track 18, sector 18 must be allocated and at least one byte must be written to it. The drive will announce an *illegal track or sector* if the first byte is not a zero or a valid track number.

The program contains a rather fancy menu. One feature it neglects to mention is that pressing the F5 key produces the letter "z" plus a number. This can be use to create file entries which may later be deleted easily.

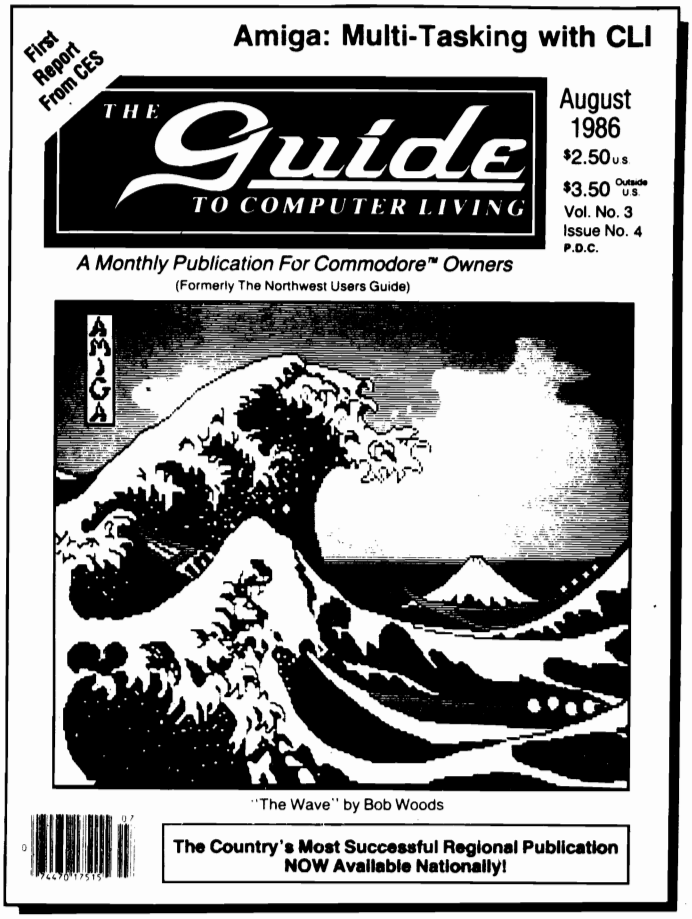
Two variables, **writover** and **filetype**, are defined at the beginning of the program. **Filetype** is the filetype of the dummy files: the default is USR, but they may also be other types, such as sequential or program files. **Writover** defines what the program does with any scratched files it finds. When **writover** is **FALSE**, the directory entries for any scratched files are appended at the end of the directory. **Writover TRUE** ignores scratched files; they are deleted from the directory. When **writover** equals 2, scratched files are replaced by dummies without changing their position in the directory. This permits you to *change a dummy file without starting the job again from the beginning.* **Writover FALSE** *is the default.* ■

If You Read Only ONE Computer Magazine A Month It SHOULD Be The Guide To Computer Living!

The unique combination of informative articles and honest reviews have made The Guide the Commodore support magazine with thousands of readers across the contry. Find out why people are saying that The Guide is the only magazine putting the "personal" back into personal computing!

Each issue of The Guide is packed with informative and understandable articles, hard-hitting reviews, and some of the best computer humor you'll find anywhere. See for yourself why readers are saying that The Guide provides more useful and reliable information than all of the other Commodore publications combined.

In the last three years The Guide has evolved from a small regional newsletter into the fastest growing Commodore magazine in the country. Don't miss this opportunity to subscribe and save 40% off the cover price.



Make checks payable to:

The Guide To Computer Living
3808 S.E. Licyntra Court
Portland, OR 97222

To Order by Phone, Call: (503) 654-5603

☐ VISA ☐ MasterCard

Card #: _____

Exp. Date: _____

Signature: _____

Phone #: _____

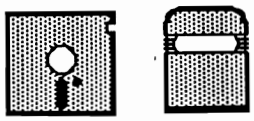
☐ **1 Year.....$18.00** (Canadian rate–$24.00 U.S.)
☐ **2 Years.....$35.00** (Canadian rate–$47.00 U.S.)
☐ **3 Years.....$48.00** (Canadian rate–$64.00 U.S.)

Name: _____

Address: _____

City, State & Zip: _____

Smart File Reader

by Captain COMAL 

Every now and then you may find an SEQ text file on a disk and wonder what is in it. The **type** command on Super Chip is a quick way to display text files. But it splits words if they fall at the end of a line. The short file reader program listed in the next column is **smart**. It knows what the maximum number of characters per line is (set at the top of the program to 39 for use with a 40 column screen). As it is now, it prints a blank line after each section of text. Remove the **PRINT** line in the **readit** procedure if the blank line is not desired.

COMAL 0.14 users should **not** type in the **file'exists** function as listed. Instead type in the function from page 44 of *COMAL Today #15*.

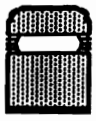
The need for this program arose at the January COMAL meeting on QLink. **Prof** mentioned that he was uploading 29 disks full of text files ... the full Bible text ... into our **COMAL Library** on QLink. You are invited to download them all if you wish! Prof compressed the files with a utility called **ARC** to save on download time. You need a copy of ARC to uncompress the files after you download them. You can download the ARC program from QLink as well. Specific instructions on how to do it are posted in our **COMAL Today** Message Board. Once uncompressed, the files are ordinary SEQ files.

We'd like to thank Prof for taking the time to compress and upload all those files. If you write any programs to use the files, we hope you will upload your programs to our library on QLink for other users to use.

To find the files, sign on to QLink and go to the **Commodore Information Network** (**CIN**). Next choose **Magazine Rack**, then **COMAL Today** and you will be in our section. Scan our Message Base to find the latest instructions. Have plenty of **formatted** blank disks ready when you go to the **COMAL Library**!

```
DIM intext$ OF 500, filename$ OF 20
maxchars:=39 // max char per line
REPEAT
  INPUT "filename (q to quit):": filename$
  IF filename$<>"q" THEN
    IF file'exists(filename$) THEN
    readit(filename$,maxchars) //wrap line
  ENDIF
UNTIL filename$="q"
//
PROC show(text$,max) CLOSED
  // max is max line length
  WHILE LEN(text$)>max DO
    pos:=max+1
    IF " " IN text$(1:pos) THEN
      WHILE text$(pos)<>" " DO pos:-1
      mark:=pos+1
    ELSE
      mark:=pos
    ENDIF
    PRINT text$(1:pos-1)
    text$:=text$(mark:LEN(text$))
  ENDWHILE
  PRINT text$
ENDPROC show
//
PROC readit(filename$,maxchars)
  OPEN FILE 2,filename$,READ
  WHILE NOT EOF(2) DO
    INPUT FILE 2: intext$
    show(intext$,maxchars)
    PRINT //blank line after each
  ENDWHILE
  CLOSE
ENDPROC readit
//
FUNC file'exists(filename$) CLOSED
  TRAP // this is the 2.0 version
    OPEN FILE 78,filename$,READ
    CLOSE FILE 78
    RETURN TRUE
  HANDLER
    CLOSE FILE 78
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists ■
```

CBase

by Russ Jensen 

Cbase is a menu driven data base program I have been working on for well over a year. It lets you create records with your choice of the number and size of fields. The display can be formatted for the screen or printer and records can be sorted as needed. Data is normally stored in **RANDOM** (REL) files, but a special feature allows it to read and write SEQ files for use with many word processors. *Cbase* is on *Today Disk #16.*

These are instructions on how to use the *cbase* data base program. After loading the program and typing **RUN** you will receive a series of prompts to follow to get started, either to create a new data base file or to manipulate an existing one. I will tell you what prompts and screen displays to expect, and how to respond to them.

Prompt: File Name

Enter the name you wish to call your file (10 characters maximum). This name will be used to keep track of all disk files associated with the file you are creating. If you want to work with an existing file, type in the name you gave it when it was first created.

Prompt: Insert Data Disk
Press Any Key to Continue

Insert the disk on which you want your new file to be written (you must use a formatted diskette), or the disk on which your existing file is located. Then hit any key such as «*space*». Other existing files on a diskette will not be harmed by creating a new file.

Prompt: New File? y or n

Y indicates you are creating a brand new file. N indicates you wish to work with an already existing file.

NEW FILES

Prompt: # of Fields

Enter the number of fields you wish your new file to have.

Prompt: Name of Field N
(N indicates a field number)

Enter the name you want to use for that field. Names should be descriptive of the field's contents.

Note: The field name should not contain more characters than specified for the length of that field or headings in reports will overlap. The field name should also be 10 characters maximum.

Prompt: Length of Field N

Enter the maximum number of characters that data in field N can have.

Note: Once a field length is specified and a file is started, it cannot be changed. Therefore, you must carefully select your field lengths so that any possible data will not exceed them. You must also take into account, however, the space that each field will take on your printed reports.

Note: These last two prompts will be repeated for the number of fields you specified in response to the Number of Fields prompt. After all field names and lengths have been entered, they will be displayed for review.

Prompt: Changes y or n

If you answer y, the entire list of field definitions you have just entered will be ignored. You will return to the Number of Fields prompt. If you type **n**, the field formats are accepted and the following displays and prompts appear:

more»

Display: Writing 'F-FOR' File

At this time a disk file is written which contains all field names, lengths, etc, information required to maintain your file.

Display: Writing 'TOPR' File

A file is now written containing the record number of the next record which would be added (1 in the case of a new file).

Prompt: Approx # of Records

Enter a guess as to how many records are going to be in your file. If you guess low, no problem, this is only used to initialize space on the disk the first time. Any time you reuse the file you are allowed to change this number upward.

This completes the *initialization procedure* for new files. Next the main menu will be displayed.

EXISTING FILES

Display: Initializing Old File

This indicates the system is being set up to use an existing file.

Display: Reading 'F-FOR' File

The format file (written when your file was initially defined) is read to give the program information it needs to maintain your file.

Display: Reading 'TOPR' File

The *top record* file, which contains the record number of the next record to be added (total number of records in the file +1) is read.

Note: This file is updated every time the cbase quit operation is performed (unless no records were added, or deleted).

Display: Total # of Records Now is N

The number N displayed is the number of records in the file.

Prompt: Change Approximate Length to

Input your new estimate as to the number of records in your file after this session.

Note: This number must always be greater than or equal to the current number previously displayed.

At this point the program will go to the main menu.

MAIN MENU OPTIONS

The following is a brief description of the options available from the main menu. These do the actual work on the data base. Each operation is fairly *user friendly* and should not give you much trouble. I will also briefly discuss a somewhat confusing point about using the program, namely printer formats.

1) ADD (Add records)

This option allows you to add records to the data base. After you have entered the various fields (as you are prompted to do) each record will be displayed and you are asked *"Any Changes?"*.

Answer **y** to correct errors, one field at a time. You will be prompted as to which field you wish to change, shown what is currently in that field, and asked for your new input. The record will again be displayed and you may then change any other field (or the same one again) until your record is correct.

Answer **n** for no changes. You return to the main menu.

more»

Answer **r** for no changes, but to remain in the add mode to enter additional records.

2) READ (Display Records)

This mode allows you to display any of the records in the data base on the screen. To use this option, however, you must know the record number of the record you wish to display. If you do not know the record number use the search/change option, explained below. After the record is displayed you are asked for *any changes*. See add above for the results of your answer (except that **r** will repeat the read mode).

3) FIELD SEARCH

This is one of the most useful modes of a data base program. You may search for and display (or print) all records which meet the search criteria you specify. You input the field number of the field you wish to search on and the value to look for in that field. When searching for your given value you are given options of two types of searches. A prompt will ask you:

Match Anywhere in field ('y' or 'n')?

If you answer **y**, a *match* will occur anytime the value string you entered occurs anywhere in the selected field of any record in your file.

If you answer **n**, a match of your entered value must occur either exactly, or the first K characters of the field must equal the K characters typed for the value. For example, if you want to search for all entries in a field starting with **a** you would use a search value of **a**. If you are searching a year field and you want all records in which the year is sometime in the 1950's your search value would be 195.

After you have entered the search field and value, you are asked if you want the results of the search sorted (on any field or combination of fields you specify). You then select either *display* (screen) or *printer* for your output. If printer is selected you must enter a *printer format* (see explanation below). If you select display, the records which meet your search criteria are displayed on the screen, one at a time. Press any key to see the next record.

4) DISPLAY

This mode displays all records in the data base on the screen. You are asked if you want your data sorted before it is displayed. You can ask for a sort on any combination of fields you specify. Each record is then displayed on the screen. To see the next record, hit any key.

5) PRINT

This is essentially the same as display, except that the output goes to the printer instead of the screen. Printing is continuous. A *printer format* must be entered before printing can begin.

6) DISPLAY FORMAT

This option allows you to look at the field format which was defined when the file was first created, the name of each field and its length.

7) SEARCH/CHANGE

This is a special combination of field search and read in which you specify the contents of a particular field of the record(s) you want to view (or change), instead of its record number. The same search options are available as in field search.

After a record is located which matches your search criteria, the contents of that record are displayed. You are asked if you want to change it as described under the read option described above. After that the search process continues until all records, which meet your criteria, have been displayed (and changed if desired).

more»

8) DELETE RECORD

You specify the record number of the record you wish deleted from your data base. The record is then displayed and you have the option of deleting it or not. If you don't know the record number of a record you wish to delete, find it by first using search/change to display the proper record along with its record number. Then use delete with that Record number.

9) QUIT

This option should **always** be used when you are finished working on your data base. It insures the proper records are written to disk and files are **CLOSED** so that the data base can be used the next time. You will be asked to confirm quit. If you say **y** the program will be properly terminated. If you say **n** you will be returned to the main menu.

10) LOAD DATA

This option allows you load data from a SEQ file into your database. The file can either be created on a word processor, which produces standard SEQ files, or from the save data option of *cbase* (see below). These files must have one field per line (each field followed by a «*return*»). Of course, the number of fields per record must match the database into which you are loading. When this option is selected, you are prompted for the name of the disk file to load from. The data is then loaded into the database. As each record is loaded, its contents (and the record number assigned by *cbase*) are displayed on the screen. When loading is complete you are returned to the main menu.

11) SAVE DATA

This option allows you to write the data from your records into a standard SEQ text file. This file can then be read by a word processor (which uses SEQ text files) or by the *cbase* load data function. You are asked if you want your file sorted before it is output. This would be very useful if your file was a mailing list which you wanted to use as input to a label making program. After the sort (if requested) is complete, the data is written to a SEQ file called *filename.save*, where *filename* is the name of your *cbase* database file. In this file the data is written with each field of each record followed by «*return*».

12) MASS CHANGE

This option allows you to store duplicate information in a particular field of all records in the database. It displays the current contents of that field for record #1 and asks you what you wish to change it to. Remember, this will change the contents of the chosen field for all records in your database.

PRINTER FORMATS

Any time you use one of the program options which requires printing, you are asked to input a *printer format*. You are prompted for each field in your data base to enter the row and column in which you wish that field to be printed. The row will be 1 if you want all fields of each record to be printed on the same line. If you wish to print your fields on more than one line you may use other rows. If you want a blank line in between records you must not specify line 1 (put your fields on lines 2 or beyond) this will leave line 1 as the blank line between records. The column number you specify is the actual printer column.

After you have specified the row and column numbers for each field in your data base, the format you specified is displayed on the screen. You have a chance to change any or all field printer formats.

more»

Note: Ignore the R to Repeat prompt option when you are asked if you wish to change the printer format (ie, either answer y or n).

Note: If you wish to not print any of your fields in a particular report, enter a column number of zero for that field(s). You must enter a non zero row number, which is either equal to or greater than the row number of the previous field.

Note: If you enter a printer format for one menu option and a format is required by use of a subsequent option, the format you previously entered will be displayed. You can then opt to reuse it for the current printout or change it.

Note: If during actual printing of data to the printer, the program determines that your current printer format is not compatible with your field definitions, printing will be halted and the message tab error will appear momentarily on the screen. You will then see the printer format displayed on the screen and you must now change it (one field at a time) until it is correct. If this happens don't forget to advance the paper in the printer to a new page since printing of the report begins at the beginning.

As soon as you have accepted a print format, you are asked if you wish to save it to disk. If you answer y, the current printer format will be saved to disk. That format will be loaded into the program the next time the program is run from the start.

CBASE - HINTS AND TIPS

1) FILE PLANNING

The field definitions for your file should be carefully chosen **before** you enter them. Once they are set, and data is put into the file, they cannot be changed without starting **all over again**. This means that all your data would have to be re-entered after you set up a new file with different field definitions. This would even be true if you only had to add **one character** to **one field** because you originally had not made that field big enough.

Note: If you ever do run into this problem there is a way out. Use the cbase save data feature to save your database to a sequential file. Then create a new database, using the same fields in the same order, but with longer field lengths. After the new database is created, use the load data function to load the data from the previous database into the new one.

Another important consideration in choosing your field lengths is how they will fit on any report you print out. You should try to make your fields fit on one print line (approximately 75 characters, including spaces between fields) whenever possible, as this will produce neat reports. But do not shortchange yourself in field length just to make everything fit on one line. *Cbase* allows printer formats with multiple lines/record and even provides multi-line headers to match. If you choose a multi-line format you must however create your printer format to give a blank line between records, if you want one.

2) RECORD NUMBERS

Cbase uses **RANDOM** (REL) files for the storage of data. This type of file requires a record number to identify each record in the file. *Cbase* assigns record numbers sequentially, staring with one for the first record entered into your data base. The only exception to this is that if you delete a record, using the delete option, the deleted record is replaced by the last record in your data base. The number of total records is decremented. Therefore, the record number of any record is also the sequence number showing when that record was entered, except when records have been deleted.

Some of the menu options, such as read, require you to know the record number of the record you wish to access. These numbers are somewhat

more»

arbitrary. The search/change option has been provided to allow you to find the number of any record to view or change, as long as the contents of at least one field are known.

The print and display options normally access data in record number order. Again, these options allow their outputs to be sorted on any field(s), thus providing alphanumerically ordered output if desired.

3) CBASE DISK FILES

Each *Cbase* data base has three primary disk files associated with it. The *data file* itself is a **RANDOM** (REL) file named *filename.data.* (*Filename* is the name you gave your data base during creation).

A second file, the top record file, is called *filename.topr*. This is a SEQ file that contains the record number of the next record which would be added to the data base (this is the total number of records plus one).

The third file, the format file is named *Filename.f-for*. It contains information which *Cbase* needs to know to maintain the data base. This is also a Seq file which contains the following information:

a. The number of fields in your data base.
b. The name of your file (*filename*).
c. The field name of field number one.
d. The length of field number one.
e. A pointer to the position of field number one in the record.

Note: These last 3 items (c, d, and e) are repeated for each field in your data base

f. The total record length
g. An end-of-file marker.

Note: The topr and f-for files described above are standard Commodore SEQ files.

In addition to the 3 files just mentioned, additional sort index files are created each time you ask that a file be sorted by a particular field(s). The names of these files are of the form *filename.s.#*, where # is the number of the particular sort field(s) that file represents. You can have many of these files depending on how many field combinations you have sorted.

These sort index files are COMAL **READ/WRITE** type SEQ files which contain a list of record numbers arranged in the order on which the data was sorted. Any time any field(s) are changed, added, or deleted, all sort index files previously generated are scratched. New sorts must be performed to replace the files.

If you request that your printer format be saved to disk, *cbase* will also create a print format file. This file is a COMAL **READ/WRITE** type SEQ file which has the file name, *filename.p.* This file contains the row and column information for each of the print fields.

If you use the save data option of *cbase*, save files will be created. These are standard SEQ files compatible with word processors, such as *Easyscript*. They have the name, *filename.save.*

4) NOTES ON SORTING

Many options of *cbase* allow you to sort your data on a given field (or combination of fields). The sorting process is somewhat slow.

When a sort is requested, the program asks you how many fields to sort the data on. After answering that prompt, you are given a display of all your field names and field numbers. You are asked to select your first sort field, second field, etc. During this selection you must use the field numbers, not the field names. Once you have selected the same number of sort fields as you originally specified, the sorting process begins.

more»

Extra Programs

Several messages are printed to the screen during a sort to indicate what is going on. You are told on which field the data is being sorted, that a sort table is being generated, and when the data is actually being sorted.

During that time, a number is displayed on the screen showing where the sorting process is at any given time. This number is the recursion level of the quicksort process. If this number stays constant for more than a few seconds, something is probably wrong. If you stop the program at that point you must close your files. To do this type **CLOSE** «*return*» after you have stopped the program using the «*run/stop*» key. This should protect your files.

When the actual sorting has been completed, a message on the screen indicates that a sort index file is being written, along with the corresponding sort index number(s).

Note: If you have added or deleted files to your database, and then had to manually stop the program, your topr file has probably not been updated. After you type CLOSE, and the red light on the disk drive has gone out, you should then type write'topr «return» before proceeding.

When sorting is complete, the message:

Do You Want to Sort on Another Field

is displayed. If you answer **n**, the program continues with the function you originally selected. If you answer **y**, the sorting process repeats asking for another field (or combination of fields) to sort on. It should be noted that these additional sorts only create the corresponding sort files for future use. The program remembers the original sort field(s) you requested sorting on for the functions (display, print, etc) in which you invoked the sort option.

Note: Sorting is done to a depth of 10 characters. ■

STAR TREK - THE DATA BASE

Last issue we presented a data base system to keep track of Doctor Who shows. Charl Phillips has turned it into a *Star Trek* data base system. A full 80 block data file is on the front side of *Today Disk 16*. This data will work with both the COMAL 0.14 and 2.0 programs (which also are on the disk).

¤¤

SHORT BATCH FILES

Ever looked with a disk editor at a batch file? I did so recently, and discovered that the ones I examined had a good many unnecessary spaces. With the program *shorten'bat'file* on *Today Disk #16*, I was able to shorten my batch files to one half the original length or better. Since I had some other problems with carriage returns in a program to make a package from a batch file, this one also removes consecutive carriage returns. - Jack Baldrige

¤¤

CUBIC SPLINES

This program by Herbert Wollman fits a cubic spline to a set of (x,y) points. The example plots a continuous sine function, given only a few points to start from. There are two versions. One allows the spacing between x values to vary. A faster one requires the spacing between x values to remain the same. This is different from the splines program on page 66 that allows the curve to cross over itself like a pretzel. In this program, there can only be one y value for any given x value. *Cubic'splines* and *uniform'splines* are on *Today Disk #16*.

¤¤

There are a few other programs on *Today Disk #16* including a mad surprise from Bill Nissley. ■

It's in the Dice

by Joel Ellis Rea

This is a general-purpose function that rolls dice! It rolls up to 7 dice, displays them in a row on the screen, and returns their sum. The dice may be any color (normal or reversed) and can be rolled selectively. The function header is:

func roll'dice(count,color,rev,ref dice$) closed

The parameters are:

count := Number of dice to roll.
color := Color to display them in.
rev := Flag for reversed display.
dice$:= A referenced string variable dimensioned **OF** at least **count**. It has two purposes. On input, any character whose **ORD** value is zero causes the corresponding die to be *rolled*. Otherwise, the **ORD** value is assigned to the die. On output, each character's **ORD** value simply equals the value of the corresponding die.

The dice are displayed using the upper/ graphics character set (that mode is automatically set). The dice take up 5 text screen lines and are **always** left-justified. Here is a sample call:

```
dummy$="";craps=roll'dice(2,2,1,dummy$)
```

The above rolls 2 dice, displays them in reversed red (color=2, rev=1), and returns the sum which is assigned to **craps**. **Dummy$** is set to the empty string, which means roll both dice.

```
dice$(2)=""; dice$(4)=""
total=roll'dice(5,6,0,dice$)
```

That displays 5 dice in normal blue (color=6, rev=0), but only 2 (dice #2 and #4) are actually re-rolled (assuming that **dice$** had a value from a previous **roll'dice** call). The sum is assigned to **total**, and the individual die values are stored as the **ORD** values of the corresponding characters of **dice$**. **Dice$(2)** and **dice$(4)** receive new values, the other characters remain the way they were before the call. This kind of a call might be found in a Yahtzee game!

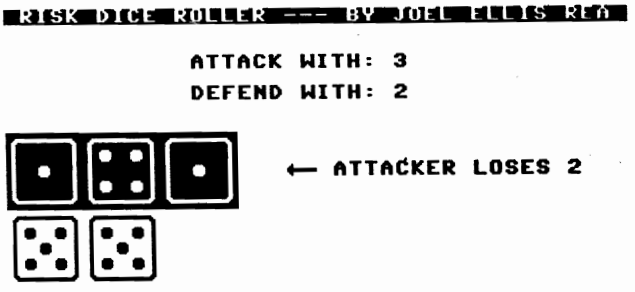
To use **roll'dice** in your own program, simply load your program, then:

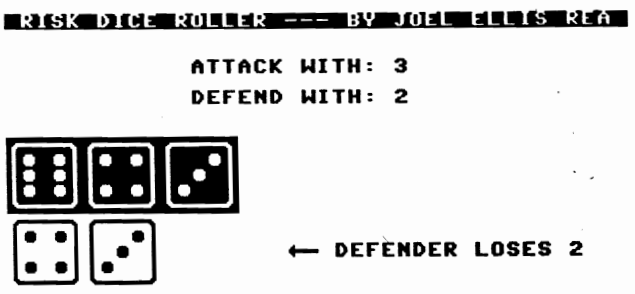
MERGE "func.roll'dice"

There is a sample program on *Today Disk #16* which illustrates **roll'dice**. It is called *risk'dice*, and is for use with the popular board game, RISK. It handles *battle* rolls, and tells how many armies the attacker and defender each lose. The dice as drawn on the screen are colored similar to the real RISK dice. The program is also a good example of how to write nearly user-proof programs!

Further Reference:

COMAL 2.0 External Procedures, COMAL Today #7, page 27
Pass an Array as a Parameter, COMAL Today #4, page 48
Parameters, COMAL Today #2, page 32 ■





Custom Listings

by Dick Klingens 

On page 222 in the *COMAL Handbook* we find a program to change the setup modes for program listing and editing. *[However, it was written for COMAL 2.0 on a Commodore PET computer with a COMBI board and won't work with the c64 COMAL 2.0 cartridge.]* The cartridge has a control location at **$c7d8,** called BORGE. The address can be used to make program listings readable by other versions of COMAL. As you perhaps know the older versions have **NEXT** instead of **ENDFOR**, and some versions require **EXEC** for procedure calls. Well, BORGE can change program listings in the way you like. The default settings for the cartridge are:

bit 0: 0 - control code as ""<ctrl>""
bit 1: 0 - no insert mode
bit 2: 0 - no NEXT in listings (but ENDFOR)
bit 3: 0 - no EXEC in procedure calls
bit 4: 0 - not used (?)
bit 5: 0 - not used (?)
bit 6: 0 - names in lower case (no upper)
bit 7: 1 - keywords in upper case

Some of the bits are influenced by COMAL commands. The 3rd bit can be set and reset by **SETEXEC+** and **SETEXEC-**; the bits 0 and 1 are both set after

```
USE system
quote'mode(TRUE)
```

The bits 6 and 7 can be set with:

```
USE system
names'in'upper'case(TRUE)
keywords'in'upper'case(FALSE)
```

If you want to configure your system, you can use the program called *customize'list* on *Today Disk #16. Customize'list* prompts you with the default settings which can be changed to meet your needs. After all the settings have been updated, **LIST** the program to see the new format already in place.

```
// Setup modes - for listing
DIM prompt$ OF 40, reply$ OF 4
PAGE
modes:=$c7d8
FOR bit:=0 TO 7 DO
  READ prompt$
  POKE modes,on'off(prompt$,bit)
ENDFOR bit
END "MODES NOW SET"
//
FUNC on'off(prompt$,bit)
  IF prompt$="skip" THEN return peek(modes)
  reply$:="> n"157""
  IF bit=7 THEN reply$(3):="y"
  PRINT prompt$+spc$(34-len(prompt$))+reply$,
  INPUT AT 0,0,1: "": reply$
  IF reply$="N" OR reply$="n" THEN
    RETURN PEEK(modes) BITAND $ff-2^bit
  ELSE
    RETURN PEEK(modes) BITOR 2^bit
  ENDIF
ENDFUNC on'off
//
DATA "CONTROL CODES listed in reverse"
DATA "QUOTE MODE and INSERT MODE on"
DATA "List the keyword EXEC"
DATA "List NEXT instead of ENDFOR"
DATA "skip","skip"
DATA "Identifiers in UPPER case"
DATA "Keywords in UPPER case" ■
```

Extended Print Using

by Robert Shingledecker

In *COMAL Today #13* someone asked if it was possible to output numbers with commas like this: 34,000.00. The standard built in **PRINT USING** does not allow this. While writing a **CLOSED** procedure to produce the commas, I decided to add a floating dollar sign.

before: **PRINT USING "#####.##":num'var**
after: **print'using("##,###.##",num'var)**

To float a dollar sign, be sure that there is at least one extra $ to the left of the number.

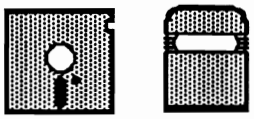
print'using("$$$,$$$.$$",num'var)

This will float a dollar sign for num'var up to 99,999.99. The format itself will be printed if the number doesn't fit properly.

The following procedure is on *Today Disk #16* with a demo program.

```
PROC print'using(fmt$,number) CLOSED
  DIM nbr$ OF 20
  float:=FALSE
  IF fmt$(1)="$" THEN float:=TRUE
  IF "." IN fmt$ THEN
    decimals:=LEN(fmt$)-("." IN fmt$)
    number:=number*(10^decimals)
  ELSE
    decimals:=0
  ENDIF
  number:=number+.5
  number:=INT(number)
  nbr$:=STR$(number)
  IF decimals THEN
    IF number=0 THEN
      FOR i:=1 TO decimals DO
        nbr$:+"0"
      ENDFOR i
    ENDIF
    IF LEN(nbr$)<decimals THEN
      tmp$:=""
      FOR i:=1 TO decimals-LEN(nbr$) DO
        tmp$:+"0"
      ENDFOR i
      tmp$:+nbr$
      nbr$:=tmp$
    ENDIF
    point:=LEN(nbr$)-decimals
    tmp$:=nbr$(1:point)
    nbr$:=tmp$+"."+nbr$(point+1:LEN(nbr$))
  ENDIF
  lf:=LEN(fmt$)
  ln:=LEN(nbr$)
  c:=0
  FOR i:=1 TO lf DO
    IF fmt$(i)="," THEN c:+1
  ENDFOR i
  IF ln>lf-c THEN
    PRINT fmt$,
  ELSE
    j:=0
    c:=0
    s:=0
    IF NOT (decimals) THEN s:=1
    FOR i:=ln TO 1 STEP -1 DO
      IF s THEN
        IF nbr$(i)<>"-" THEN c:+1
        IF c=4 THEN
          fmt$(lf-j):=","
          j:+1
          c:=1
        ENDIF
      ENDIF
      fmt$(lf-j):=nbr$(i)
      IF nbr$(i)="." THEN s:=1
      j:+1
    ENDFOR i
    FOR i:=lf-j TO 1 STEP -1 DO
      IF float THEN
        fmt$(i):="$"
        float:=FALSE
      ELSE
        fmt$(i):=" "
      ENDIF
    ENDFOR i
    PRINT fmt$,
  ENDIF
ENDPROC print'using ■
```

Floating Point Error

by David Stidolph 

In the last issue of *COMAL Today*, we published a **POKE** that would change the method of drawing on the Hi-Res graphics screen. It flips the status of a pixel, rather than setting it (so that an <u>on</u> bit is turned <u>off</u>, and an <u>off</u> bit is turned <u>on</u>). As a demonstration for this, I wrote a short COMAL 2.0 graphics program that drew a series of eight boxes rotated about a point. I intended to draw boxes in one pass and erase them in the next. I was quite surprised to find that after several passes, some of the pixels never got erased. The group of boxes slowly floated up towards the top of the screen, leaving a trail behind.

The answer turned out to be quite simple. It seems that COMAL 2.0 carries out all its graphic functions using floating point numbers. A small roundoff error accumulated over a period of time, preventing the drawing from ending on the same point it started from.

What was more surprising was that when I tried this in COMAL 0.14 (with a different **POKE** to change drawing to flipping), the pattern stayed in its original position. Is COMAL 0.14 better than COMAL 2.0 with floating point math? I don't know, but it makes for an interesting discussion. *[See page COMAL Today #15, page 23 for another example of the difference in the roundoff error between the different versions of COMAL.]*

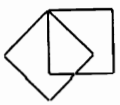
<u>COMAL 2.0 Program</u>:

```
USE graphics // initialize graphics
graphicscreen(0)
moveto(160,100) // move to center
POKE $c462,$5d // flip mode to draw
//
REPEAT
  FOR y:=1 TO 8 DO
    box
    right(45)
  ENDFOR y
UNTIL TRUE=FALSE
//
PROC box
  FOR x:=1 TO 4 DO
    right(90)
    forward(50)
  ENDFOR x
ENDPROC box
```

<u>COMAL 0.14 program</u>:

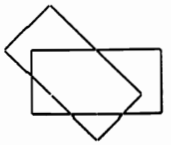
```
setgraphic (0)
moveto 160,100
hideturtle
poke 27669,93
//
repeat
 for y:=1 to 8 do
  box
  right (45)
 endfor y
until true=false
//
proc box
 for x:=1 to 4 do
  right (90)
  forward (50)
 endfor x
endproc box ■
```

STEP 1

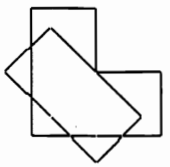


STEP 2

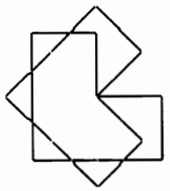
STEP 3



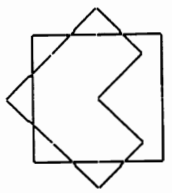
STEP 4



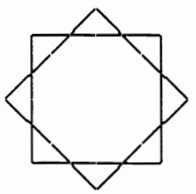
STEP 5



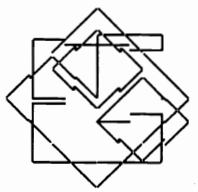
STEP 6



STEP 7



STEP 8



Read and Run System

by David Stidolph and Colin Thompson

The idea of the **Read and Run System** is rather simple. Programs and their articles share space on a disk, eliminating the need for separate written documentation. One special file contains the names of the text files to read and programs to run. This file, *names.dat*, also contains other information explained below:

```
0«
0«
13«
1«
14«
Read and Run Disk«
June«
Read Run Disk for June«
Read Me«
*end*«
Run Me«
```

actual word processor screen

```
0« <--background color
0« <--border color
13« <--text color for prompts
1« <--color to use for menu
14« <--highlighting color
Read and Run Disk« <--disk name
June« <--month
Read Run Disk for June« <--header
Read Me« <--article name
         <--more article names
         <--can go here
*end*« <--put after last article
Run Me« <--program name
        <--more program names can
        <--go here
```

same screen with comments

The **Read and Run** *menu* program learns what text files and programs are on the disk by reading the file *names.dat*. This file contains the necessary file names, plus information for the screen colors, and header information such as the date the disk was created. One example of its format is shown above.

The article and program titles (from the end of the *names.dat* file) are used as filenames as well as menu choices. Program file names are identical to the titles, but the suffix .txt is tacked onto the article title to become the filename (all upper case letters are also converted to lower case). For instance, the article Read Me matches the file read me.txt.

The main menu gives you the choice of reading articles, running programs, or printing all the articles complete with a table of contents. You may also exit the program into COMAL.

The article text files should be Commodore ASCII SEQ files. Each line of text is limited to 37 characters and must end with a carriage return. The file is printed in one column on the screen, or in two columns on the printer. Note: some word processors allow you to set margins and insert control codes into the text, but then give an option to *print* the file to disk as it would appear on the printer. This file would have the correct margins and carriage returns, but none of the control codes. If your word processor has this option, it is the one you should use to create your **Read and Run** files.

To produce your own **Read and Run** disk, you first need to format the disk with the title and id you want to use. Copy the COMAL 0.14 system onto this disk (use the top 9 files from the 0.14 side of *Today Disk #16*). Then gather your programs and articles. Note, your programs must be COMAL 0.14 programs; you can't run a BASIC program from a COMAL menu. The articles must follow the above conventions such as the 37 character per line limit. File names must be entirely in lower case and end with *.txt*.

more»

Those who plan to donate a **Read and Run** disk of their own creation to a users group should be aware that not everyone is familiar with COMAL. It is not necessary to know COMAL to use the **Read and Run** system. COMAL is included on the disk. All the user needs to do is follow the menus. The files *comal article* and *info.txt* describe COMAL to beginners. They are part of the 9 system files you copied to your disk.

Your disk is ready, except for the *names.dat* file. You should type this in a word processor and save it as a normal SEQ type text file (*«ctrl»*-z in *PaperClip*, or f1-f using *Easy Script*).

Today Disk #16 contains the 0.14 version of the **Read and Run** system. The 2.0 system is included on *Shareware disk #1*.

```
dim'variables // by Colin Thompson
initialize'system
repeat
 menu
 repeat
  move'bar(men$,5+3)
 until selected
 execute'option(row)
until false
//
proc dim'variables
 max:=9 // #articles
 dim disk'name$ of 16
 dim month$ of 12
 dim name$ of 20
 dim articles$(max) of 12
 dim programs$(max) of 16
 dim print'or'view$(3) of 20
 dim text$ of 38
 dim info'text$ of 80
 max'lines:=50
 dim left'column$(max'lines) of 37
 dim option$ of 1
 dim s$ of 24
 dim header$ of 40
 dim men$(5) of 19
 men$(1):="Read COMAL Articles"
 men$(2):="Run COMAL Programs"
 men$(3):="Learn About COMAL"
 men$(4):="Print ALL Articles"
 men$(5):="Quit This Menu"
endproc dim'variables
//
proc initialize'system
 disk'name$:="Proper Disk"
 promptcolor:=7
 menucolor:=5
 open'file("names.dat")
 input file 78: back'
 input file 78: bor'
 input file 78: promptcolor
 input file 78: menucolor
 input file 78: extracolor
 input file 78: disk'name$
 input file 78: month$
 input file 78: header$
 background back'
 border bor'
 pencolor promptcolor
 num'programs:=0; num'articles:=0
 repeat
  input file 78: name$
  if name$<>"*end*" then
   num'articles:+1
   articles$(num'articles):=name$
  endif
 until name$="*end*"
 while not eof(78) do
  num'programs:+1
  input file 78: programs$(num'programs)
 endwhile
 num'articles:+1
 articles$(num'articles):="Quit Menu"
 num'programs:+1
 programs$(num'programs)=articles$(num'articles)
 close
 select output "ds:"
 print'or'view$(1):="Read on Screen"
 print'or'view$(2):="Print to Printer"
 print'or'view$(3):="Return to Menu"
 s$:=status$
 row:=4; new'row:=4; selected:=false
endproc initialize'system
```

more»

```
//
proc open'file(filename$)
 if file'exists(filename$) then
  open file 78,filename$,read
 else
  error
  open'file(filename$)
 endif
endproc open'file
//
proc menu
 page
 title(disk'name$,month$)
 for i:=1 to 5 do print tab(5),men$(i)
 pencolor menucolor
 for i:=1 to 7 do separator(chr$(192),39)
 cursor(15,1)
 pencolor promptcolor
 print tab(7),"COMAL Newsletter on Disk"
 for x:=1 to 7 do print
 print "Use the cursor up/down";
 print "key to choose a"
 print "submenu. Then press";
 print "return to execute."
endproc menu
//
proc move'bar(ref array$(),max'row)
 selected:=false; move'it:=false
 case direction of
 when -1
  if row>4 then
   new'row:=row-1; move'it:=true
  endif
 when +1
  if row<max'row then
   new'row:=row+1; move'it:=true
  else
   new'row:=4; move'it:=true
  endif
 when 2
  selected:=true
 otherwise
 endcase
 //
 if move'it then
  cursor(row,3)
  print chr$(146);
  print ;array$(row-3),tab(28),
  cursor(new'row,3)
  print chr$(18);
  print ;array$(new'row-3),tab(28),
  row:=new'row
 else
  cursor(row,3)
  print chr$(18);
  print ;array$(row-3),tab(28)
 endif
endproc move'bar
//
func direction
 option$:=key$
 case option$ of
 when chr$(17)
  return +1
 when chr$(145)
  return -1
 when chr$(13),chr$(13+128)
  return 2
 otherwise
  return 0
 endcase
endfunc direction
//
proc execute'option(row)
 case row-3 of
 when 5
  exit'to'comal
 when 4
  print'all
 when 3
  read'a'file("comal article")
 when 1
  display'files(articles$,num'articles)
  title("Feature Articles",month$)
  select'file(articles$,num'articles,0)
 when 2
  page; box
  print chr$(29);chr$(18);
  print "COMAL Programs on this Disk";
  print tab(37)
  cursor(4,1)
  pencolor promptcolor
```

more»

```
   for i:=1 to num'programs do
    print tab(5),programs$(i)
   endfor i
   for i:=1 to (24-num'programs)/3 do
    separator(chr$(192),39)
   endfor i
   row:=4
   select'file(programs$,num'programs,1)
  otherwise
  endcase
endproc execute'option
//
func file'exists(prog$)
 open file 78,prog$,read
 s$:=status$
 close file 78
 return s$(1:2)="00"
endfunc file'exists
//
proc error
 page; box
 print chr$(29);"ERROR:";chr$(18),s$
 cursor(12,5)
 print "Put";disk'name$;"in the drive"
 cursor(13,5)
 print "and press any key to continue"
 wait
endproc error
//
proc cursor(row,col) closed
 poke 211,col-1
 poke 209,(1024+(row-1)*40) mod 256
 poke 210,(1024+(row-1)*40) div 256
 poke 214,row-1
endproc cursor
//
proc shift'wait
 pencolor extracolor
 while not peek(653) and not eof' do
  print "Press SHIFT to read or";
  print "^ to quit",chr$(17+128)
  print tab(37),chr$(17+128)
  if esc or key$="^" then eof':=true
 endwhile
 pencolor promptcolor
endproc shift'wait
//
proc file'to'print(name$)
 for x:=1 to len(name$) do
  if name$(x)>chr$(127) then
   name$(x):=chr$(ord(name$(x))-128)
  endif
 endfor x
 pencolor menucolor
 if file'exists(name$) then
  open file 5,name$,unit 8,read
  open file 255,"",unit 4,7,write
  select output "lp:"
  repeat
   for i:=1 to 40 do left'column$(i):=""
   eof':=false; full:=false
   stopped:=false
   read'left'column; print'a'page
  until eof'
  close
  select output "ds:"
  pass "i0"
 else
  error
 endif
endproc file'to'print
//
proc check'for'eof
 if esc then eof':=true
 if key$="^" then eof':=true
 if line'count=max'lines then full:=true
 stopped:=eof'
endproc check'for'eof
//
proc read'left'column
 page'num:+1; line'count:=0
 repeat
  if not eof(5) then
   line'count:+1
   input file 5: left'column$(line'count)
   check'for'eof
  else
   eof':=true
  endif
 until eof' or full
endproc read'left'column
//
```

more»

```
proc print'a'page
 page'header
 if eof' then
  for i:=1 to line'count do
   print left'column$(i)
  endfor i
 else
  print'line:=0
  repeat
   if not eof(5) then
    print'line:+1
    input file 5: text$
    print left'column$(print'line),
    print tab(41),text$
    check'for'eof
   else
    eof':=true
    for i:=print'line+1 to max'lines do
     print left'column$(i)
    endfor i
   endif
  until print'line=max'lines or eof'
 endif
 page'footer
endproc print'a'page
//
proc page'header
 separator("-",79)
 print header$,tab(71),"PAGE";page'num
 print "MONTH:";month$;"1986",tab(44),
 print "Article Filename:";name$
 separator("-",79)
 print
 print
endproc page'header
//
proc page'footer
 for i:=1 to 52-line'count do print
 for i:=1 to 20 do print "=",
 print ;;"For more information";
 print "please contact";;
 for i:=1 to 19 do print "=",
 print chr$(13)
 print "COMAL Users Group, USA Limited"
 print "6041 Monona Drive"
 print "Madison, WI 53716"
 for i:=1 to 3 do print
endproc page'footer
//
proc wait
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc wait
//
proc exit'to'comal
 page; box
 print chr$(29);"Welcome to disk";
 print "loaded COMAL 0.14 !!",chr$(17)
 print chr$(18),"You may load";
 print "these COMAL 0.14 programs:"
 print
 pencolor extracolor
 for i:=1 to num'programs-1 do
  print "chain """,programs$(i),""""
 endfor i
 pencolor promptcolor
 print
 print "Or, you may enter BASIC like this:"
 print
 pencolor extracolor
 print "basic"
 pencolor promptcolor
 trap esc+
 end
endproc exit'to'comal
//
proc page
 print chr$(147),chr$(14),
endproc page
//
proc read'a'file(name$)
 page
 pencolor promptcolor
 cursor(4,1)
 for i:=1 to 3 do
  print tab(5),print'or'view$(i)
 endfor i
 row:=4
 repeat
  move'bar(print'or'view$,6)
 until selected
 trap esc-
```

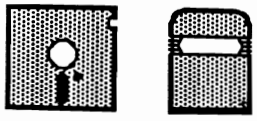
more»

```
 case row-3 of
 when 1
  file'to'screen(name$)
 when 2
  line'up'paper; page'num:=0
  file'to'print(name$)
 otherwise
 endcase
 while esc do null
 trap esc+
endproc read'a'file
//
proc display'files(ref array$(),max'files)
 page
 pencolor promptcolor
 cursor(4,1)
 for x:=1 to max'files do
  print tab(5),array$(x)
 endfor x
 row:=4
 pencolor menucolor
 for i:=1 to (24-max'files)/3 do
  separator(chr$(192),39)
 endfor i
 pencolor promptcolor
endproc display'files
//
proc select'file(ref name$(),max,z)
 exit:=false
 cursor(23,1)
 pencolor promptcolor
 if z then
  print chr$(18);"Highlight the";
  print "program you want to run "
  print chr$(18);"and press return.";
  print "The program will be "
  print chr$(18);"""CHAINed"".";
  print "(Automatic LOAD and RUN).";;
 else
  print
  print chr$(18);"Highlight the";
  print "article you want to     "
  print chr$(18);"read and press";
  print "the RETURN key         ",
 endif
 repeat
  move'bar(name$,max+3)
 until selected
 if z then
  text$:=name$(row-3)
 else
  text$:=name$(row-3)+".txt"
 endif
 for x:=1 to len(text$) do
  v:=ord(text$(x:x))
  if v>128 then text$(x:x):=chr$(v-128)
 endfor x
 if row-3<max then
  if file'exists(text$) then
   if z then
    chain text$
   else
    read'a'file(text$)
   endif
  else
   error
  endif
 endif
endproc select'file
//
proc title(string$,month$)
 box
 print chr$(29);chr$(18);string$;
 print "for";month$,tab(38)
 print
endproc title
//
proc separator(char$,num)
 pencolor menucolor
 if num<50 then print
 for t#:=1 to num do print char$,
 print
 pencolor promptcolor
endproc separator
//
proc line'up'paper
 page
 pencolor promptcolor
 cursor(12,3)
 print "Align the paper near the";
 print "top of form."
 cursor(14,5)
```

more»

```
 print "Press any key to";
 print "begin printing."
 wait; page; cursor(12,4)
 pencolor extracolor
 print "Press the ^ key to";
 print "stop printing."
 print "(Printing stops at the";
 print "end of the page)"
endproc line'up'paper
//
proc file'to'screen(name$)
 pencolor promptcolor
 if file'exists(name$) then
  open file 5,name$,unit 8,read
  print chr$(147),chr$(14)
  cursor(25,1)
  eof':=false
  repeat
   if not eof(5) then
    input file 5: text$
    print ;text$
    shift'wait
   else
    eof':=true
   endif
  until eof'
 else
  error
 endif
 close'files
endproc file'to'screen
//
proc close'files
 close file 5
 select output "ds:"
 pencolor promptcolor
 print chr$(18);"Press any key. "
 wait; page
endproc close'files
//
proc box
 pencolor menucolor
 print chr$(19),
 print chr$(176),
 for x:=2 to 38 do print chr$(192),
 print chr$(174)
 print chr$(221),tab(39),chr$(221)
 print chr$(173),
 for x:=2 to 38 do print chr$(192),
 print chr$(189)
 print chr$(19)
 pencolor promptcolor
endproc box
//
proc print'all
 line'up'paper
 page'num:=1
 file'to'print("comal article")
 if not stopped then
  for i2:=1 to num'articles-1 do
   poke 49152+i2,page'num+1
   file'to'print(articles$(i2)+".txt")
   if stopped then i2:=99
  endfor i2
  if i2<99 then
   page'num:=1
   name$:="Table of Contents"
   open file 255,"",unit 4,7,read
   select output "lp:"
   page'header
   print "COMAL Article",tab(41);"2"
   for x:=1 to num'articles-1 do
    print articles$(x),tab(41),
    print using "##": peek(49152+x)
   endfor x
   print
   line'count:=x+1
   if file'exists("info.txt") then
    open file 2,"info.txt",read
    while not eof(2) do
     input file 2: info'text$
     print info'text$
     line'count:+1
    endwhile
    close file 2
   endif
   page'footer
   select output "ds:"
   pass "i0"
  endif
 endif
endproc print'all ■
```

Learning Subtraction

by Len Lindsay 

If you have kids just learning how to subtract, your Commodore 64 or 128 can help. I have written a short program that picks subtraction problems that fit whatever range you desire. It can adapt to ask only subtraction questions with an answer of 0, 1, 2, or 3. Or it can set up problems with answers ranging from zero to nine. It is set up for only one digit answers. After each problem, the top of the screen is updated showing the number of problems presented, how many right answers were given, as well as the average number of seconds needed for each answer. A nice touch was to print this information in reverse field if the previous answer was incorrect. I do **not** have a wrong answer produce an awful noise or print WRONG on the screen.

Let's look at how the program works. The main program is very simple:

```
repeat
  erase'screen
  init
  problems
until yes'no("Do you want more")=false
```

The first line is **REPEAT**. It tells us that we will be repeating something. COMAL automatically indents the lines inside the **REPEAT** loop so it is easy to see what we are repeating.

First we erase the screen. Then we initialize the system. Finally we do the problems. That is easy to see from the program listing.

OK, I cheated. **erase'screen** is not a built in keyword. I defined it myself. COMAL lets us define any new words we want. Look at the very end of the program listing. You will see my definition of **erase'screen**. All it does is print a **CHR$(147)**. This is the way you clear the screen on a Commodore computer. Yes, we could have just put the line **PRINT CHR$(147)**, directly in the **REPEAT** loop. But then the listing would not be as readable.

To initialize the system we call the procedure named **init**. You have noticed that routines in COMAL are called by name rather than by line number. COMAL uses line numbers to help us edit the program, but you never refer to the line numbers in the program itself. Instead, you give your routines names. These routines are called procedures and functions. **Erase'screen** and **init** are procedures.

Init sets some variables that adapt the program to your specifications. I chose to keep the program interactive, thus it will ask you some questions when you run it. Notice that long variable names are used, not just letters like **X** and Y. By using long variable names, you can keep your program readable. Months later you can look at the program and remember what you were doing.

The program needs to know how many spaces to indent the problem when it prints it on the screen, as well as the maximum number to use for the top and bottom numbers. It also allows you to set a maximum answer and how many problems you want to try. Look at how I get those numbers. I could have used simple **INPUT** statements just like in BASIC. But I chose to create a function to get the numbers. This way I can present a default answer. Now, you can quickly answer each question by simply hitting the «*return*» key. This automatically gives the default answer for each question. Look at the first line in the **init** procedure:

```
indent'amount:=get'num("Indent Problem
Amount",19) //wrap line
```

This line is setting the variable **indent'amount** to the value that you want. We simply provide my **get'num** function with a prompt and a number to use as the default (19 in this case). The function will print your prompt on the screen, a question

more»

mark, and blink the cursor on the suggested answer. You may type in a different answer, or merely hit the «*return*» key to accept the suggestion. The function then RETURNs the number you entered.

After the system is initialized, the program presents the **problems**. Notice how the **problems** module includes lines like **get'problem** and **show'problem**. This shows how you can break a large part of your program into several smaller pieces. Then write each smaller piece, one at a time. For example, **get'problem** does what its name suggests: it gets the top and bottom numbers for the next problem.

Hopefully you can look over the rest of the program listing and see what is happening. You may notice several lines that include **PRINT USING**. This built in COMAL feature will format numbers to the number of decimal places you desire. You also might notice another built-in feature of COMAL: the **IN** keyword. It searches through a string for you. The third line inside the **yes'no** function checks if the reply was either a "Y" or a "y" (shifted or not).

Next issue I will present another COMAL program for your use in learning subtraction (it will require a printer). It prints a practice page of subtraction problems. Use it for practice or for a time test. It can print multiple copies, so several people can take the same test, and provides an answer sheet as well. Till then, speak COMAL. It gives you control of your computer.

Follow these steps to type in the program:

1) Start the COMAL system.

2) Clear out any previous programs:

NEW

3) Let COMAL provide the line numbers:

AUTO

4) Type in the program listed below.

5) Stop AUTO mode:

COMAL 0.14: Hit «*return*» key twice
COMAL 2.0: Hit «*stop*» key

Then to save your program:

SAVE "SUBTRACT"

The ",8" that BASIC requires is not used. Remember, you must delete the old file first if you use the same name for the new file (the drive number 0: is required by COMAL 0.14):

DELETE "0:SUBTRACT"

To get your program back again:

LOAD "SUBTRACT"

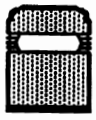
If you don't wish to type in the program, it is available on *Today Disk #16*.

```
dim reply$ of 1
repeat
  erase'screen
  init
  problems
until yes'no("Do you want more")=false
//
proc init
  indent'amount:=get'num("Indent Problem
  Amount",19) // wrap line
  max'top:=get'num("Max top number",19)
  max'bot:=get'num("Max bottom number",10)
  max'answer:=get'num("Max answer number",9)
  max'problems:=get'num("Number of problems"
  ,9) // wrap line
  correct:=0; avg'time:=0
endproc init
```

more»

```
//
func yes'no(question$) closed
  dim reply$ of 1
  input question$+"? ": reply$
  if reply$="Y" or reply$="y" then
    return true
  else
    return false
  endif
endfunc yes'no
//
proc problems
  zero'time
  header(0) // for startup
  for problem:=1 to max'problems do
    get'problem
    show'problem
    get'reply
    update
  endfor problem
endproc problems
//
func get'num(prompt$,default)
  print prompt$;default
  print chr$(145), // cursor up
  input prompt$+"?": number
  return number
endfunc get'num
//
proc header(prob'num)
  erase'screen
  print using "Prob:###": prob'num;
  print using " Right:###": correct;
  print using "  Avg Time:##.# secs": avg'time
  for blanks:=1 to 5 do print
endproc header
//
proc show'problem
  indent
  print using " ##": problem'top
  indent
  print using "-##": problem'bot
  indent
  print "==="
endproc show'problem
//
proc get'reply
  repeat
    reply$:=key$
  until reply$ in "0123456789"
  guess:=ord(reply$)-48
endproc get'reply
//
proc update
  avg'time:=jiffies/60/problem
  if guess=problem'top-problem'bot then
    correct:+1
  else
    print chr$(18), // reverse on
  endif
  header(problem)
endproc update
//
proc indent
  print tab(indent'amount),
endproc indent
//
proc get'problem
  problem'top:=rnd(1,max'top)
  low:=problem'top-max'answer
  if low<0 then low:=0
  high:=problem'top
  if high>max'bot then high:=max'bot
  problem'bot:=rnd(low,high)
endproc get'problem
//
proc zero'time
  poke 160,0
  poke 161,0
  poke 162,0
endproc zero'time
//
func jiffies
  j=256*256*peek(160)+256*peek(161)+peek(162)
  return j
endfunc jiffies
//
proc erase'screen
  print chr$(147),
endproc erase'screen ■
```

The Game of Nim



by Jack Baldrige

The game of nim, and others similar to it, have received a good bit of attention from mathematicians. At any stage of the game, each player knows what his opponent knows. Also, the permissable moves at any time depend on pattern of counters and not on what moves were made to get there.

Nim has been completely analyzed, so I had to impose certain rules so the computer won't win every game. In the COMAL game on *Today Disk #16*, the number of stones in each row is determined by the number of rows; the computer plays second and whoever takes the last stone wins the game. With these constraints, you will always win the game if you make no errors. When playing at the higher levels, you'll have an interesting time avoiding all errors.

For the beginner, the program has a help feature, which allows you to see whether taking a certain number of stones from a row will leave you with a safe position. You may also use **PROC help** to see if the computer's position is safe, by asking the result of taking no stones from a row. (If a position is safe before you take any stones, it will be unsafe for you after you've taken one or more stones.)

It's worth noting that one way to determine whether a position is safe is to express the number of stones in each row as a binary number, then add the stones in all the rows together without carrying. If the result is even (or zero), the position is safe. The add without carry can be done in COMAL with the **BITXOR** operator. As usual, COMAL makes it easy.

If you want to learn more about nim and similar games, you'll find it discussed in an interesting way in the book, *Wheels, Life and Other Mathematical Amusements* by Martin Gardner.

```
DIM st#(9)
intro(startfl#,total#,cs,ps)
REPEAT
  LOOP
    IF startfl# THEN
      start'play(st#(),startfl#,total#)
      you'play(st#(),startfl#,total#)
    ELSE
      you'play(st#(),startfl#,total#)
    ENDIF
    playdone(1,st#(),total#,done#,startfl#,cs,ps)
    IF startfl# THEN EXIT
    c64'plays(st#(),total#)
    playdone(2,st#(),total#,done#,startfl#,cs,ps)
    IF startfl# THEN EXIT
  ENDLOOP
UNTIL done#=TRUE
textcolors(6,6,1)
END ""147"Thanks for Playing."
//
PROC you'play(REF st#(),REF startfl#,total#)
CLOSED // wrap line
  IMPORT print'board,help
  LOOP
    print'board(st#(),total#)
    REPEAT
      print at 13+total#,1: " Take: How Many",
      PRINT " Stones?"
      INPUT AT 13+total#,25: "": q$
      IF q$="help" THEN help(st#(),total#)
    UNTIL LEN(q$)<=2
    s#:=VAL(q$)
    PRINT AT 14+total#,1: "       From What",
    PRINT " Row?",SPC$(15)
    INPUT AT 14+total#,25,2: "": p#
    IF p#>0 AND p#<=total# THEN
      IF s#>0 AND s#<=st#(p#) THEN EXIT
    ENDIF
    PRINT AT 13+total#,1: SPC$(39)
    print at 14+total#,1: " Try again.";SPC$(28)
    FOR k#:=1 TO 2400 DO NULL
  ENDLOOP
  st#(p#):-s#
ENDPROC you'play
//
PROC print'board(s#(),total#) CLOSED
```

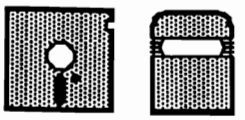
more»

```
  PRINT AT 10,1: SPC$(40*total#-1)
  FOR k#:=1 TO total# DO
    PRINT AT 9+k#,4: "Row";k#,":";
    FOR j#:=1 TO s#(k#) DO
      PRINT "+",
      IF j# MOD 5=0 THEN PRINT " ",
    ENDFOR j#
    PRINT AT 9+k#,36: USING "[##]": s#(k#)
  ENDFOR k#
  PRINT AT 13+total#,1: SPC$(79)
ENDPROC print'board
//
PROC c64'plays(REF st#(),total#) CLOSED
  RANDOMIZE
  DIM test#(9)
  IMPORT safe
  LOOP
    void#:=0; f#:=RND(0,total#-1)
    FOR k#:=1 TO total# DO
      IF st#(k#)=0 THEN
        void#:+1
      ELSE
        decrement#:=st#(k#); r#:=k#
      ENDIF
    ENDFOR k#
    IF void#=total#-1 THEN
      st#(r#):=0
      EXIT
    ENDIF
    FOR j#:=f# TO f#+total# DO
      m#:=j# MOD total#; m#:+total#*(m#=0)
      FOR k#:=1 TO total# DO test#(k#):=st#(k#)
      WHILE test#(m#) DO
        test#(m#):-1
        IF safe(test#(),total#) THEN
          r#:=m#; decrement#:=st#(m#)-test#(m#)
          st#(m#):=test#(m#)
          EXIT
        ENDIF
      ENDWHILE
    ENDFOR j#
    r#:=RND(1,total#)
    IF st#(r#) THEN
      decrement#:=RND(1,st#(r#))
      IF decrement#>1 THEN decrement#:-1
      st#(r#):-decrement#
      EXIT
    ENDIF
  ENDLOOP
  PRINT AT 3,4: "Computer takes";decrement#;
  PRINT "from Row";r#,".   "
ENDPROC c64'plays
//
PROC playdone(pl,REF st#(),REF t#,REF done#,
REF stfl#,REF cs,REF ps) closed // wrap line
  IMPORT print'board
  quitflag:=1
  FOR j#:=1 TO t# DO
    IF st#(j#) THEN quitflag:=0
  ENDFOR j#
  IF quitflag=0 THEN
    done#:=FALSE
  ELSE
    IF pl-1 THEN
      cs:+1
      PRINT AT 5,2:""158""18"",SPC$(12),
      PRINT "Computer Wins!",SPC$(12),""159""
    ELSE
      ps:+1
      PRINT AT 3,1: SPC$(39)
      PRINT AT 5,2: ""158""18"",SPC$(15),
      PRINT "You Win!",SPC$(15),""159""
    ENDIF
    stfl#:=1
    PRINT AT 25,8: SPC$(25),
    print'board(st#(),t#)
    print at 13+t#,2:"Score is You:";ps,",c64:"; cs
    PRINT AT 15+t#,2: ""158"Want to play",
    PRINT " another game?(Y/N) Y",
    INPUT AT 15+t#,34,1: "": opt$
    IF opt$="Y" OR opt$="y" THEN
      done#:=FALSE
    ELSE
      done#:=TRUE
    ENDIF
  ENDIF
ENDPROC playdone
//
PROC intro(REF fl#,REF t#,REF cs,REF ps)
CLOSED // wrap line
  fl#:=1; t#:=3; ps:=0; cs:=0
  PAGE
```

more»

```
  PRINT AT 1,16: "** NIM **"
  INPUT AT 5,2: "Instructions?(y/n) n"157"": opt$
  IF opt$="N" OR opt$="n" THEN RETURN
  PAGE
  PRINT AT 1,4: "The computer starts by",
  PRINT " asking what"
  PRINT " level of play you want.  Level 2 is"
  PRINT " easiest."
  print at 5,4:"You and the computer take turns"
  print " removing stones. A number of stones"
  print " may be removed from one row only in"
  print " each turn. The person removing the"
  PRINT " last stone from the board is the"
  PRINT " winner."
  PRINT AT 12,4: "By pressing F1 when asked",
  PRINT " how many"
  PRINT " stones you wish to remove, you can"
  PRINT " find whether the board would be safe"
  PRINT " for you if you if you removed a"
  PRINT " given number of stones from a given"
  PRINT " row."
  PRINT AT 19,4: "If you play correctly at",
  PRINT " each turn,"
  PRINT " you will win.  If you make one"
  PRINT " mistake, the C64 will win.  You"
  PRINT " play first."
  PRINT AT 25,6: ""18"  Press Any Key to",
  PRINT " Continue  "146"",
  WHILE KEY$<>"" DO NULL
  WHILE KEY$="" DO NULL
ENDPROC intro
//
PROC start'play(REF st#(),REF start'playfl#,
REF total#) // wrap line
  PAGE
  startfl#:=0
  USE system
  defkey(11,"help"13"")
  textcolors(0,0,3)
  REPEAT
    PRINT AT 1,16: "** NIM **"
    PRINT AT 5,2: "Level of play?(2-9)",
    PRINT total#,SPC$(15)
    INPUT AT 5,22,1: "": total#
  UNTIL total#>=2 AND total#<=9
  FOR k#:=1 TO total# DO st#(k#):=1+2*k#
  PRINT AT 5,2: "  Your Turn:",SPC$(25)
  PRINT AT 7,1: "";
  FOR k#:=1 TO 38 DO PRINT "*",
  PRINT AT 8,2: ""18"          12345 67890",
  PRINT " 12345 6789       "
  PRINT AT 11+total#,1: "";
  FOR k#:=1 TO 38 DO PRINT "*",
  PRINT AT 25,10: ""18" Press F1 for",
  PRINT "Help "146"",
ENDPROC start'play
//
FUNC safe(s#(),total#) CLOSED
  v#:=0
  FOR r:=1 TO total# DO v#:=v# BITXOR s#(r)
  RETURN NOT v#
ENDFUNC safe
//
PROC help(st#(),total#) CLOSED
  IMPORT safe
  PRINT AT 14+total#,1: SPC$(30)
  FOR k#:=1 TO 500 DO NULL
  DIM t#(9)
  REPEAT
    FOR k#:=1 TO total# DO t#(k#):=st#(k#)
    PRINT AT 13+total#,1: ""155" How Many",
    PRINT " Stones?",SPC$(20)
    INPUT AT 13+total#,19,2: "": s#
    PRINT AT 14+total#,1: SPC$(35)
    PRINT AT 13+total#,1: "          From",
    PRINT " Row?",SPC$(20)
    INPUT AT 13+total#,19,2: "": r#
    IF r#<=total# THEN t#(r#):-s#; goflag:=1
    IF s#>st#(r#) OR r#>total# THEN
      PRINT AT 14+total#,1: " Try",
      PRINT " again.",SPC$(20)
      goflag:=0
    ELIF safe(t#(),total#) THEN
      sf$:="Safe"
    ELSE
      sf$:="Unsafe"
    ENDIF
  UNTIL goflag
  PRINT AT 14+total#,1: " ";s#;"From",
  PRINT " Row";r#;"is";sf$,"."159""
ENDPROC help ■
```

Tiny Disk Directories

by Gerald K. Hobart 

If your printer supports condensed printing (17 characters per inch), subscripts/ superscripts and variable line spacing you can use this program to print a disk directory so small that an entire 144-entry directory will fit on the disk jacket. It was written for the *Gemini SG-10 printer/ Cardco +G interface* combination but can be easily modified for other printers which support these functions.

To read the directory into memory, I used Ray Carter's **read'dir** procedure. Next, I used Jack Baldrige's **blocks'free** function to get the number of blocks free. *(Half the work was done for me before I started.)*

One problem I encountered is that my printer prints Commodore graphic symbols in a non-standard width, ruining the appearance of the printout. Since graphic symbols are generally used only to create partitions to make the directory more readable, I wrote a procedure, **remove'graphic'symbols**, which changes all Commodore graphic symbols to periods. This solved the problem.

Procedure **tinyprint** sends control codes to the printer to turn on superscript printing, set line spacing to 1/12 inch, and select condensed printing mode.

Procedure **normalprint** sends codes to return the printer to its normal printing mode.

The control codes sent by these two procedures work with my *Gemini/Cardco* combination; they may need to be modified to work on other systems. The code is well commented so you can easily find the code that needs to be changed. In particular, if you don't get condensed printing (17 characters per inch), try changing the **CHR$(20)** to **CHR$(15)**.

If your printer doesn't support superscripts or variable line spacing, but does have condensed printing, you can still make some fairly compact listings with this program.

If you have a need to write a lot of text in a small space, you can use **tinyprint** and **normalprint** in your own programs.

Versions are included on *Today Disk #16* for both COMAL 2.0 and 0.14.

Further reference:

Custom Directories, COMAL Today #14, page 37
Blocks Free, COMAL Today #13, page 30
Fast Dir Revisited, COMAL Today #12, page 50

```
// delete"0:dirprnt'tiny"
// save  "0:dirprnt'tiny"
// by Gerald K. Hobart
//
setup
repeat
  print chr$(147)
  print "Insert Disk and Ready Printer"
  print "Hit Any Key to Begin..."
  while key$=chr$(0) do null
  print "2,3 or 4 Columns? :"
  inkey(a$,"234")
  read'dir(d$,id$,n$,t$,m,b#)
  c:=blocks'free
  remove'graphic'symbls
  open file 255,"",unit 4,7,write
  select output "lp:"
  print
  print tab(5),d$;id$;
  tinyprint
  if a$="2" then
    print2col
  elif a$="3" then
    print3col
  else
    print4col
  endif
  print using " ### BLOCKS FREE": c
```

more»

```
  print cr$,cr$,cr$
  normalprint
  select output "ds:"
  print "Another Disk (Y/N)? "
  inkey(a$,"yYnN")
until a$ in "nN"
stop
//
proc setup
  disk'get'init
  bit'init
  dim d$ of 16, id$ of 2, a$ of 1
  dim n$(144) of 16, cr$ of 2
  dim t$(144) of 4, b#(144)
  cr$:=chr$(13)
endproc setup
//
proc read'dir(ref d$,ref id$,ref name$(),ref
t$(),ref num,ref b#()) closed // wrap line
  dim block$ of 32, junk$ of 1
  num:=0
  pass "i0"
  open file 78,"$0",unit 8,read
  for ii#:=1 to 142 do junk$:=chr$(disk'get
  (78,eof78)) // wrap line
  block$:=""
  for ii#:=1 to 27 do block$:=block$+chr$
  (disk'get(78,eof78)) // wrap line
  strip(block$(1:16),d$)
  strip(block$(19:20),id$)
  for ii#:=1 to 85 do junk$:=chr$(disk'get
  (78,eof78)) // wrap line
  repeat
  block$:=""
  for ii#:=1 to 30 do block$:=block$+chr$(disk'
  get(78,eof78)) // wrap line
    if ord(block$(4)) and ord(block$(1)) mod 16
    num:+1 then // wrap line
      case ord(block$(1)) mod 16 of
      when 1
        t$(num):="seq"
      when 2
        t$(num):="prg"
      when 3
        t$(num):="usr"
      when 4
        t$(num):="rel"
      otherwise
        t$(num):="***"
      endcase
      if not bitand(ord(block$(1)),128) then
      t$(num):=t$(num)+"<" // wrap line
      name$(num):=block$(4:19)
      b#(num):=ord(block$(29))
      b#(num):+ord(block$(30))*256
    endif
    if (num mod 8)<>0 then
      for i:=1 to 2 do junk$:=chr$(disk'get
      (78,eof78)) // wrap line
    endif
  until eof78
  close file 78
endproc read'dir
//
proc strip(string$,ref result$)
  if chr$(160) in string$ then
    result$:=string$(1:(chr$(160) in string$)-1)
  else
    result$:=string$
  endif
endproc strip
//
proc tinyprint closed
  open file 4,"",unit 4,7,write
  print file 4: chr$(27),chr$(211),chr$(0),
  // turn on superscript
  print file 4: chr$(27),chr$(193),chr$(6),
  //line spacing to 1/12 inch
  print file 4: chr$(20) // select condensed print
  close file 4
endproc tinyprint
//
proc normalprint closed
  open file 4,"",unit 4,7,write
  print file 4: chr$(27),"@",
  // return printer to normal mode
  close file 4
endproc normalprint
//
proc remove'graphic'symbls
  for ii#:=1 to m do
    for jj#:=1 to len(n$(ii#)) do
```

more»

```
      x:=ord(n$(ii#)(jj#))
      if x=160 then
        n$(ii#)(jj#):=" "
      elif x>90 and x<193 or x>218 then
        n$(ii#)(jj#):="."
      endif
    endfor jj#
  endfor ii#
endproc remove'graphic'symbls
//
proc print2col
  m2:=int(m/2)+m mod 2
  print
  for i:=1 to m2 do
    print using "###": b#(i);
    print n$(i);
    print t$(i),tab(26),
    j:=i+m2
    if j<=m then
      print using "###": b#(j);
      print n$(j);
      print t$(j)
    else
      print
    endif
  endfor i
endproc print2col
//
proc print3col
  m3:=int(m/3+.67)
  print
  for i:=1 to m3 do
    print using "###": b#(i);
    print n$(i);
    print t$(i),tab(26),
    j:=i+m3
    print using "###": b#(j);
    print n$(j);
    print t$(j),tab(51),
    j:=i+2*m3
    if j<=m then
      print using "###": b#(j);
      print n$(j);
      print t$(j)
    else
      print
    endif
  endfor i
endproc print3col
//
proc print4col
  m4:=int(m/4+.76)
  print
  for i:=1 to m4 do
    print using "###": b#(i);
    print n$(i);
    print t$(i)(1),tab(24),
    j:=i+m4
    print using "###": b#(j);
    print n$(j);
    print t$(j)(1),tab(47),
    j:=i+2*m4
    print using "###": b#(j);
    print n$(j);
    print t$(j)(1),tab(70),
    j:=i+3*m4
    if j<=m then
      print using "###": b#(j);
      print n$(j);
      print t$(j)(1)
    else
      print
    endif
  endfor i
endproc print4col
//
func blocks'free closed
  dim dummy$ of 1
  pass "i0"
  open file 78,"$0",unit 8,read
  for i:=1 to 2 do dummy$:=chr$(disk'get
  (78,eof78)) // wrap line
  blocks#:=0
  for x#:=1 to 35 do
    sectors:=disk'get(78,eof78)
    for i:=1 to 3 do dummy$:=chr$(disk'get
    (78,eof78)) // wrap line
    if x#<>18 then blocks#:+sectors
  endfor x#
  close file 78
  return blocks#
endfunc blocks'free
```

more»

Read Directory

This COMAL 0.14 program can be used to read a disk directory. The program prints the directory to the screen or to a printer. The program uses the **read'block** and **disk'get'init** procedures from *COMAL Today #15*. The information needed for the **get'entries** procedure is contained in the article, *Unscratching Files* in *COMAL Today #13*. It isn't hard to write a new program when most of the work has been done for you. All you need to do find out what procedures are available, and then practice putting them together.

This directory program is not as versatile as *directory'probe* from *Today Disk #13*, but it has a few advantages of its own. If you only need a print out of a disk directory, this program is shorter and easier to use. This program may also be adapted for other uses. All the directory entries are stored in an array. You could write a routine to see if a particular program is on a disk by comparing file names to the one you are looking for. You could write routines to sort the names or to list all files of a given type. You could even store the file names on a disk to keep track of which programs are on which disks. Of course most of this work has been done for you too. It is in the book/disk set, *Captain COMAL Gets Organized*.

The following program is on *Today Disk #16*.

```
dim d$ of 16, id$ of 2, out$ of 1
dim a$(144) of 16, t$(144) of 4
dir(d$,id$,c,a$,t$)
print chr$(147),chr$(14)
input "Printer output (y/n): n"+chr$(157): out$
if out$="y" or out$="Y" then
 select output "lp:"
endif
print "Disk:",d$;"ID:",id$
print
for x:=1 to c do
 print using "###]": x;
 print a$(x),tab(25),t$(x)
endfor x
select output "ds:"
```

more»

```
//
proc inkey(ref c$,valid$) closed
  while key$<>chr$(0) do null
  repeat
    c$:=key$
  until c$ in valid$
endproc inkey
//
proc disk'get'init closed
  for loc:=2024 to 2039 do
    read value
    poke loc,value
  endfor loc
  data 0,162,0,32,198,255,32,207
  data 255,141,232,7,32,204,255,96
endproc disk'get'init
//
func disk'get(file'num,ref f'end) closed
  poke 2026,file'num
  sys 2025
  f'end:=peek(144)
  return peek(2024)
endfunc disk'get
//
proc bit'init closed
  for i:=702 to 733 do
    read a
    poke i,a
  endfor i
  //
  data 0,0
  data 173,190,2,45,191,2,141,190,2,96
  data 173,190,2,13,191,2,141,190,2,96
  data 173,190,2,77,191,2,141,190,2,96
endproc bit'init
//
func bitand(i,j) closed
  poke 702,i
  poke 703,j
  sys 704
  return peek(702)
endfunc bitand ■
```

```
      comal - gkh t6
 3 ed'disk'title2.0 prg   1 func.bitand      seq
14 dirprint'tiny    prg   1 func.bitor       seq
 1 ---------------- usr   1 func.bitxor      seq
 5 ed'disk'title.14 prg   2 txt.ed'dsk'title seq
16 dirprnt'tiny .14 prg   9 txt.dirprint'tny seq
 2 proc.bit'init    seq   4 txt.bit function seq
605 BLOCKS FREE
```

```
//
proc dir(ref disk$,ref id$,ref count,ref entry$(),
ref type$()) closed // wrap line
 disk'get'init(1000)
 dim block$ of 258, ds$ of 2, name$ of 16
 count:=0; sector:=1; track:=18
 pass "i0"
 ds$:=status$
 if ds$="00" then
  f'state:=read'block(18,0,block$)
  disk$:=block$(145:162)
  id$:=block$(163:164)
  strip(disk$)
  repeat
   f'state:=read'block(track,sector,block$)
   get'entries
  until sector>18 or f'state
 endif
endproc dir
//
proc get'entries
 track:=ord(block$(1))
 sector:=ord(block$(2))
 for pos:=3 to 227 step 32 do
  if block$(pos)>chr$(127) then
   count:+1
   case ord(block$(pos)) mod 16 of
   when 0
    type$(count):="del"
   when 1
    type$(count):="seq"
   when 2
    type$(count):="prg"
   when 3
    type$(count):="usr"
   when 4
    type$(count):="rel"
   otherwise
    type$(count):="***"
    if (ord(block$(pos)) mod 128) div 64 then
     type$(count):=type$(count)+"<"
    endif
   endcase
   entry$(count):=block$(pos+3:pos+19)
   strip(entry$(count))
  endif
 endfor pos
endproc get'entries
//
proc strip(ref text$)
 mark:=chr$(160) in text$
 if mark then text$:=text$(1:mark-1)
endproc strip
//
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 ■
```

Two Disk Utilities

DISK'EDITOR 

by Phyrne Bacon

The COMAL 2.0 version of *disk'editor* has all of the features of the COMAL 0.14 *disk'editor*, and some additional features as well. The program contains complete help screens so I will only mention the improvements here. The 0.14 version is described in *COMAL Today #13* and is included on its matching disk. The 2.0 version is on *Today Disk #16*.

TRACK AND SECTOR

The current track and sector are given in decimal followed by the corresponding hex; for example, *track 18 12 sector 1 01*; here 18=$12 and 1=$01.

PRINTING

The first time the printer is used, the user is asked to give the secondary command for the printer.

PRINTING THE BLOCK (@)

@ sends the current block to the printer. The information printed is the same as that displayed on the screen (there is a space between the two half-blocks) except that each line is preceded by the position number in hex of the first byte in that line.

PRINTING THE DISK NAME (m)

M prints the disk name and disk id.

DISPLAYING THE DISK NAME (*)

* displays the disk name and disk id on the screen. Press any key to continue.

PRINTING THE BLOCK CHAIN (q)

If the cursor is on a track number, and there is a sector number in the next byte, pressing **q** will cause the block chain beginning at that track/sector to be printed on the printer. If track/sector are the track/sector of a directory entry, the filename of the entry with its length and type are printed before the block chain. Scratched files are indicated by --- or (0). Non-standard filetype numbers are listed in parentheses. If a directory file is not the same length as the length listed in the directory, *wrong length, file is xx blocks long* is printed.

DISPLAYING THE BLOCK CHAIN (y)

Y is the same as **q** except that the chain is displayed on the screen.

TOGGLE POSITION (g)

The position of the cursor is listed in decimal as a number between 0 and 255. **G** changes this number to hex, or back to decimal again.

Further reference:

Disk Editor, COMAL Today #13, page 28

¤¤

DIRECTORY'PROBE

by Phyrne Bacon

Directory'probe 2.0 finds the starting track and sector for all files on a disk, even scratched files. It can optionally follow the tracks and sectors through to the end of the file. The information can be printed on the screen or on a printer. It can be useful when used together with the disk editor to alter, fix, or unscratch files. See *COMAL Today #13* for details about the COMAL 0.14 version of this program. ■

Super Chip Notes

Dear Sirs: I recently purchased a Super Chip to add to my COMAL Cartridge. My main reason for purchasing the Super Chip was to speed up disk reading. I have the disk-based version of the **Rabbit** on the *Today Disk #12*, but was unable to use it effectively because of the inability to use **Rabbit** with a printer connected.

In my testing of the Super Chip version of the **Rabbit**, I have been unable to obtain any increase in the speed with which files are read under program control. The speed-up does occur during program loading and execution of the type command. All other Super Chip commands I have tested appear to work as documented.

The test program on *Today Disk #16* creates a test file and reads it twice, once with **Rabbit** disabled, and once with it enabled. The time required to read the file is reported each time, using the **gettime$** function of the **System** package.

The results of my testing are as follows (for all tests, the printer is disconnected from the system). Having reset the computer, I run the test program with Super Chip installed in the COMAL Cartridge:

```
RUN "test'rabbit"
Normal: read'file:   0:00:58.1   type:   0:01:13.5
Rabbit: read'file:   0:01:00.2   type:   0:00:44.4
```

Resetting the computer, I then **LINK** the disk version of **Rabbit** and run the test program again:

```
LINK "pkg.rabbit"
RUN "test'rabbit"
Normal: read'file:   0:00:58.1   type:   0:01:13.5
Rabbit: read'file:   0:00:45.2   type:   0:00:41.9
```

Clearly, the Super Chip version of **Rabbit** does speed up **type** but not **INPUT FILE**. Is there an explanation? -Thomas E Bishop, Walnut Port, PA

[We aren't sure why Super Chip (and Super Chip on Disk) Rabbit does not speed up all disk reads. However, it does have the advantage of working with a printer. If you don't need to use a printer, you can still LINK in the Today Disk #12 Rabbit to obtain the faster reads.]

```
DIM text$ OF 80
USE system
USE files
USE rabbit
setfast(FALSE)
IF NOT file'exists("test.data") THEN
  PRINT "Creating test file"
  write'file
ENDIF
settime("0")
read'file // type("test.data")
PRINT "Normal time: ",gettime$
USE rabbit
setfast(TRUE)
settime("0")
read'file // type("test.data")
PRINT "Rabbit time: ",gettime$
END
//
PROC write'file
  DELETE "test.data"
  OPEN FILE 2,"test.data",WRITE
  FOR i:=1 TO 1000 DO
    text$:=STR$(i)
    PRINT FILE 2: text$
  ENDFOR i
  CLOSE FILE 2
ENDPROC write'file
//
PROC read'file
  OPEN FILE 2,"test.data",READ
  WHILE NOT EOF(2) DO
    INPUT FILE 2: text$
    PRINT text$
  ENDWHILE
  CLOSE FILE 2
ENDPROC read'file ■
```

Sorting Routines

by Susan Long

Every programmer at some time or another needs a routine to sort a list of numbers or words into numerical or alphabetical order. The easiest routine to do this is called a *bubble sort*. It is also very inefficient, because it requires a comparison between every pair of items in the list. For longer lists, this takes increasingly more time, even in COMAL. Fortunately, there are other routines available which are much faster, particularly for lists greater than 50 to 80 items. The program *demo/sorting* on *Today Disk #16* demonstrates five different sorting algorithms: bubble sort, heap sort, shell sort, insertion sort, and quicksort. Each of these is a separate procedure, which may be used independently in an applications program. The program demonstrates these procedures using a list of 40 flower names to be sorted alphabetically. Of course, each procedure can be easily modified to sort numbers instead of strings. Just change the string arrays and variables to numerical arrays and variables.

The algorithms are not easy to follow, even in COMAL. They include nested counting loops, recursion, and complex mixtures of comparison, exchange, and reorganization of variables. Those included in the program illustrate three basic types of sorting algorithms: bubblesort and quicksort are exchange methods, which interchange pairs of items if they are out of order; insertion sort and shell sort are insertion methods, which take each item and put it into its place among the others; heapsort is a selection method, which selects out items starting with the largest and putting each one in its place at a time.

The choice of which procedure to use depends on the situation. When programming from memory with a small number of items, a programmer might choose to use a bubble sort, since it is easy to code. Yet other methods, particularly heap sort, shell sort, and quicksort, are much faster and don't take up much more memory. When working with many items, 1000 for example, quicksort is on average faster than shell sort or heap sort. However, there are cases where quicksort is slower than the others, including paradoxically the case where the list to be sorted is almost in order already.

Further reference:

Sorting it Out, COMAL Today #10, page 18

```
// by Susan Long
DIM num(16), array$(40) OF 16
DIM sorted$(40) OF 16, name$ OF 20
instructions
readdata
demonstrate("bubble")
demonstrate("shell")
demonstrate("insert")
demonstrate("heap")
demonstrate("quick")
PRINT "Press shift to see the sorted names."
WHILE PEEK(653)=0 DO NULL
shownames
// main program ends here
//
DATA "loosestrife","trillium","daffodil","rose"
DATA "iris","colubine","bugloss","aster"
DATA "hollyhock","baby's'breath","zinnia"
DATA "lily","portulaca","flax","astilbe"
DATA "dustymiller","campanula","pink","vinca"
DATA "foxglove","gentian","violet","lupine"
DATA "verbena","pimpernel","daisy","poppy"
DATA "begonia","cleome","cosmos","dahlia"
DATA "gazania","gaillardia","hibiscus"
DATA "lantana","lobelia","impatiens","petunia"
DATA "primrose","delphinium"
//
// procedures begin
PROC readdata
  FOR count:=1 TO 40 DO
    READ array$(count)
  ENDFOR count
ENDPROC readdata
//
```

more»

```
PROC demonstrate(name$)
  TIME 0
  PRINT "Now doing ",name$
  CASE name$ OF
  WHEN "bubble"
    bubblesort(array$(),sorted$(),40)
  WHEN "shell"
    shellsort(array$(),sorted$(),40)
  WHEN "insert"
    insertsort(array$(),sorted$(),40)
  WHEN "heap"
    heapsort(array$(),sorted$(),40)
  WHEN "quick"
    quicksort(array$(),sorted$(),40)
  ENDCASE
  now:=TIME
  PRINT INT(now/60)," seconds for ",name$
  PRINT
ENDPROC demonstrate
//
PROC shownames
  PAGE
  FOR count:=1 TO 20 DO
    PRINT sorted$(count)
  ENDFOR count
  PRINT ""19"",
  FOR count:=21 TO 40 DO
    PRINT AT 0,21: sorted$(count)
  ENDFOR count
ENDPROC shownames
//
PROC instructions
  PAGE
  PRINT TAB(10),"Sorting Routines"
  PRINT "Written by Susan Long, Rumson, NJ"
  PRINT "Based on algorithms from the book:"
  PRINT "'Sorting and Searching' by D. Knuth"
  PRINT "These routines will sort a set of"
  PRINT "words into alphabetical order."
  PRINT "The routines are: bubblesort"
  PRINT "                  shellsort"
  PRINT "                  insertionsort"
  PRINT "                  heapsort"
  PRINT "                  quicksort*"
  PRINT "To demonstrate these procedures here"
  PRINT "I have used a list of 40 flower names."
  PRINT "For each routine,the time required to"
  PRINT "sort the names will be shown."
  PRINT "The times might be relatively"
  PRINT "different for a longer list."
  PRINT
  PRINT "Press shift to continue."
  WHILE PEEK(653)=0 DO NULL
  PAGE
ENDPROC instructions
//
PROC bubblesort(array$(),REF sorted$(),number)
CLOSED // wrap line
  DIM temp$ OF 20 // size
  FOR countj:=number TO 2 STEP -1 DO
    FOR counti:=2 TO countj DO
      IF array$(counti)<array$(counti-1) THEN
        temp$:=array$(counti)
        array$(counti):=array$(counti-1)
        array$(counti-1):=temp$
      ENDIF
    ENDFOR counti
  ENDFOR countj
  FOR count:=1 TO number DO
    sorted$(count):=array$(count)
  ENDFOR count
ENDPROC bubblesort
//
PROC shellsort(array$(),REF sorted$(),n) closed
  interval:=n
  REPEAT
    interval:=INT(interval/2)
    diff:=n-interval
    FOR j:=1 TO diff DO
      item1:=j
      switch
    ENDFOR j
  UNTIL interval=0
  FOR count:=1 TO n DO
    sorted$(count):=array$(count)
  ENDFOR count
  //
  PROC switch
    item2:=item1+interval
    IF array$(item1)>array$(item2) THEN
      temp$:=array$(item1)
      array$(item1):=array$(item2)
```

more»

```
      array$(item2):=temp$
      item1:=item1-interval
      IF item1>=1 THEN switch
    ENDIF
  ENDPROC switch
  //
ENDPROC shellsort
//
PROC insertsort(a$(),REF sorted$(),n) CLOSED
  FOR j:=2 TO n DO
    FOR i:=j-1 TO 1 STEP -1 DO
      IF (a$(j)<a$(i+1) AND a$(j)>=a$(i)) THEN
        temp$:=a$(j)
        FOR c:=j TO i+2 STEP -1 DO
          a$(c):=a$(c-1)
        ENDFOR c
        a$(i+1):=temp$
      ELIF a$(j)<a$(1) THEN
        temp$:=a$(j)
        FOR c:=j TO 2 STEP -1 DO
          a$(c):=a$(c-1)
        ENDFOR c
        a$(1):=temp$
      ENDIF
    ENDFOR i
  ENDFOR j
  FOR count:=1 TO n DO
    sorted$(count):=a$(count)
  ENDFOR count
ENDPROC insertsort
//
PROC quicksort(array$(),REF sorted$(),n)
CLOSED // wrap line
  // this routine from 0.14 disk
  partition(1,n,1,n)
  FOR count:=1 TO n DO
    sorted$(count):=array$(count)
  ENDFOR count
  //
  PROC partition(left,right,i,j)
    midval$:=array$(INT(left+right)/2)
    REPEAT
      WHILE midval$>array$(i) DO i:+1
      WHILE midval$<array$(j) DO j:-1
      IF i<=j THEN
        temp$:=array$(i)
        array$(i):=array$(j)
        array$(j):=temp$
        i:+1
        j:-1
      ENDIF
    UNTIL i>j
    IF left<j THEN partition(left,j,left,j)
    IF i<right THEN partition(i,right,i,right)
  ENDPROC partition
  //
ENDPROC quicksort
//
PROC heapsort(array$(),REF sorted$(),n) closed
  FOR left:=INT(n/2) TO 1 STEP -1 DO
    sift(left,array$(left),n)
  ENDFOR left
  // now have a heap
  FOR r:=n-1 TO 1 STEP -1 DO
    temp$:=array$(r+1)
    array$(r+1):=array$(1)
    sift(1,temp$,r)
  ENDFOR r
  FOR count:=1 TO n DO
    sorted$(count):=array$(count)
  ENDFOR count
  //
  PROC sift(i,k$,r)
    j:=i+i
    IF j>r THEN
      array$(i):=k$
    ELSE // j=r or j<r
      IF j<r THEN
        IF array$(j)<array$(j+1) THEN j:+1
      ENDIF
      IF k$>=array$(j) THEN
        array$(i):=k$
      ELSE
        array$(i):=array$(j)
        i:=j
        sift(i,k$,r)
      ENDIF
    ENDIF
  ENDPROC sift
  //
ENDPROC heapsort ■
```

Sort Package

by Robert Ross 

This package on *Today Disk #16* provides various options for sorting single dimension arrays using a partition sort method. The element indexing routines are used by the floating point, signed integer, and string value compare routines. The string sorts permit comparisons according to designated character ranges within the string elements, either left-to-right or right-to-left. The values of characters for string comparisons may be redefined in any order. Implemented functions permit embedding two byte signed integer or five byte floating point numbers into strings for sorting by embedded numeric values.

While **quicksort** in Super Chip was only for left justified string sorts, this package is over twice as long and has taken up to 35% longer to sort identical string material. However, it uses less space on the COMAL stack. The «*run/stop*» key is checked at the beginning of each partition; if depressed, the sort is terminated regardless of whether **ESC** is trapped, and trappable error 9 is reported. This package uses space at $02a7 and various zero page locations including $55 and $fb-$ff for temporary storage.

Each sort call requires at least three parameters. The first is a reference array and the next two (**ae1#** and **ae2#**) give the range of elements to be sorted in the array. If **ae1#** is less than **ae2#**, the elements are sorted in ascending order; otherwise, the elements are sorted in descending order. The next two parameters in some of the procedures, **sc1#** and **sc2#**, specify a substring for the comparisons. When **sc1#** is less than **sc2#**, the comparison is left-to-right; otherwise, the comparison is right-to-left. Trappable error 5 *value out of range* is reported if **ae1#**, **ae2#**, **sc1#** or **sc2#** exceed the dimensioned values for the array. To redefine character precedence for string comparisons, the last parameter, **xl$**, is defined. The **CHR$** value from the character in the array being sorted is used as an index to the character in **xl$** that is used for sorting comparisons.

An annotated list of the procedure and function headers in this package follows:

PROC fp'sort(REF real(),ae1#,ae2#)

Sorts part or all of a single dimension array of real numbers in ascending or descending order.

PROC int'sort(REF int(),ae1#,ae2#)

Sorts part or all of a single dimension array of signed integers.

PROC lj'sort(REF str(),ae1#,ae2#)

Sorts part or all of a single dimension array of strings using a left-to-right comparison. This is the way two strings are compared in COMAL.

PROC rj'sort(REF str(),ae1#,ae2#)

Sorts based upon a right-to-left comparison.

PROC lj'lsort(REF str(),ae1#,ae2#)
PROC rj'lsort(REF str(),ae1#,ae2#)

These are similar to the preceding two procedures, but the actual lengths of the two strings are compared first. Shorter strings compare less than longer strings. The **CHR$** values are checked only if the two string elements are the same length.

PROC str'sort'fp(REF str(),ae1#,ae2#,sc1#)

Sc1# gives the location within each string where the first of five bytes representing a floating point value should be. The string array is sorted according to the values of embedded floating point numbers. If **sc1#** to **sc1#+4** are not defined in any string element accessed, trappable error 4 *substring error* is reported.

more»

PROC str'sort'int(REF str(),ae1#,ae2#,sc1#)

Sc1# gives the location within each string where the two bytes representing a signed integer value start. The string array is sorted according to the embedded integer values. If sc1# to sc1#+1 are not defined in any string element, trappable error 4 *substring error* is reported.

FUNC str'sort(REF str(),ae1#,ae2#,sc1#,sc2#)

Sorts a single dimension string array on a designated substring. Sc1#<sc2# gives a left-to-right sort; sc1#>sc2# gives a right-to-left sort. 1 for **TRUE** is returned if every element compared is defined in the entire designated substring. 0 for **FALSE** is returned if at least one string element was not defined for the entire substring designated.

FUNC str'lsort(REF str(),ae1#,ae2#,sc1#,sc2#)

The same as the preceding sort, except the lengths of the string elements are checked first.

func xl'sort(ref str(),ae1#,ae2#,sc1#,sc2#,ref xl$)

The same as **FUNC** str'sort() but the character precedence may be defined to other than the **CHR$** values. Xl$ is the *translation string*: the **CHR$** value of the string character is used to access a character position in the translation string which should be the **CHR$** value of the redefined character precedence. 1 for **TRUE** is returned if all string elements are defined for the whole specified substring and if xl$ is not the null string: "".

For meaningful sorting, xl$ must be defined from character position 1 (containing the new **CHR$** value for ""0"") to the position of the largest character value to be encountered in the sort +1: e.g., xl$(1:256) to redefine **CHR$** values of 0 to 255. If the **ORD** values of all characters to be sorted will be < 128, xl$ needs to be defined only from 1 to 128. If the **ORD** values to sorted range from 48 to 127, xl$ still must be defined from 1 to 128.

func xl'lsort(ref str(),ae1#,ae2#,sc1#,sc2#, ref xl$) // *wrap line*

The same as **FUNC** xl'sort() but the string element lengths are compared first.

FUNC dimlen(REF str)

Returns the dimensioned length of the specified string element. If a$(3,3,3) is dimensioned to 80, the value returned for dimlen(a$(1,2,3)) will be 80. If b$ was used without being dimensioned, dimlen(b$) will return 40.

FUNC str'fp(str)

Returns the floating point value for a given five byte string. Reports trappable error 4 *substring error* if the string length is not five bytes.

FUNC fp'str$(real)

Returns a five byte string for a given floating point number.

FUNC str'int#(str)

Returns the signed integer value for a given two byte string. Reports trappable error 4 *substring error* if the string length is not two bytes.

FUNC int'str$(int)

Returns a two byte string for a signed integer

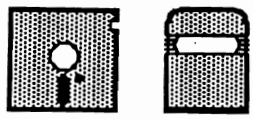
FUNC version'mlw'sorts$

Returns the version number.

Further Reference:

Super Chip Notes, COMAL Today #15, page 74
Sorting It Out, COMAL Today #10, page 18 ■

Connect the Dots



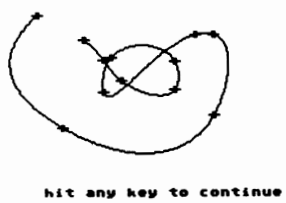
For those of you interested in this program without needing to know its history, *smooth'curve* is an advanced connect the dots program. It draws a line between all the points entered by the user, in the same order as they were entered. However, instead of drawing a straight line between the points, it uses an advanced mathematical technique know as cubic splines to draw as smooth a curve as possible. To demonstrate this, a fun drawing program is on *Today Disk #16*. Use the cursor keys to move a *pointer* around the graphics screen. Press «*return*» for each point in the path to be drawn. When all points are entered, press «*space*» to start drawing the curve.

Smooth'curve was originally a Fortran program that was translated to Pascal and then to COMAL. The Pascal version, along with a 6 page article with pictures, is in *PC Tech Journal*, August 1986. Those of you with access to that magazine may be surprised at how close the COMAL and Pascal listings actually are. There are only two major differences. First, the Pascal version has all the procedures listed before the main section of the program. COMAL can be written this way, but most COMAL programmers prefer to put the main section of the program before the procedures. The second main difference is in the names of the keywords used in the graphics commands.

COMAL 0.14 has two more differences from the Pascal version. COMAL 0.14 doesn't allow for nested procedures or the passing of arrays by value. Therefore, procedures nested in the Pascal version have been moved outside in the COMAL 0.14 version. The procedure headers were also changed to make all arrays passed by reference. These two changes were not needed in COMAL 2.0. Even with these minor changes, the structure of both the COMAL and Pascal versions of *smooth'curve* are the same.

Further Reference:

Smooth Curves, by Michael A. Covington, page 110, August 1986, PC Tech Journal, The World Trade Center, Suite 211, Baltimore, MD 21202, $29.97 per year subscription.




```
// This program is adapted from a Pascal
// program which appeared in PC Tech Journal,
// August 1986.
//
// This program draws a smooth curve
// connecting points marked on the screen by
// the user. The points do not have to define a
// function; internally, both x and y are
// functions of T, the approximate distance
// traveled as the curve moves around the
// screen. We are using 320*200 graphics.
//
dimensions
make'cross
repeat
 get'points
 fit'curve
 show'curve
 settext
 print chr$(14) // set lower case
 input "again? ": reply$
until reply$ in "Nn"
//
proc dimensions
 arraysize:=50 // max number of known points
 splinesize:=200 // number of points that are
 // calculated to draw the curve
 dim x(arraysize), y(arraysize), t(arraysize)
 dim a(arraysize), b(arraysize), c(arraysize)
 dim calcx(splinesize), calcy(splinesize)
 dim calct(splinesize)
```

more»

```
 dim cross$ of 64
 dim reply$ of 1
 dim spc40$ of 40
 spc40$(1:40):=""
 dim number$ of 3
endproc dimensions
//
proc make'cross
 border (15)
 background (0)
 pencolor (7)
 for c1:=1 to 64 do cross$(c1):=chr$(0)
 cross$(1):=chr$(28)
 cross$(4):=chr$(28)
 cross$(7:8):=chr$(227)+chr$(128)
 cross$(10):=chr$(28)
 cross$(13):=chr$(28)
 setgraphic 0
 hideturtle
 define 1,cross$
 identify 1,1
 spritepos 1,156,102
 spritecolor 1,7
 spritesize 1,0,0
endproc make'cross
//
proc get'points
 // Allows the user to move a cursor around the
 // screen and mark points. X, Y, and, T are
 // recorded for each point.
 settext
 print chr$(147) // page
 print chr$(14) // set lower case
 print "Move cursor with crsr keys."
 print "Mark point with the return key."
 print "Press space when finished."
 print
 print "You must mark at least 4 points;"
 print "space will not work until you do."
 print
 print "You can only mark"; arraysize; "points."
 print "You cannot mark the same point twice"
 print "unless you have marked a different"
 print "point in between."
 print
 print "Erroneous input will result in a beep."
 print
 input "Press return to continue: ": reply$
 setgraphic 0
 clear
 hideturtle
 identify 1,1
 pencolor (7)
 plottext 83,9,"point=1   x=160 y=100"
 plottext 27,1,"  return to mark; space to quit"
 ix#:=160
 iy#:=100
 count#:=0
 reply$:=""
 while (count#<4) or (reply$<>chr$(32)) do
  pencolor (7)
  str(count#+1,number$)
  plottext 131,9,number$
  str(ix#,number$)
  plottext 179,9,number$
  str(iy#,number$)
  plottext 227,9,number$
  graphics'cursor(ix#,iy#,1)
  repeat
   reply$:=key$
  until reply$<>chr$(0)
  case ord(reply$) of
  when 145 // up
   iy#:+1
  when 17 // down
   iy#:-1
  when 157 // left
   ix#:-1
  when 29 // right
   ix#:+1
  when 13 // return
   if same'point'again or (count#=arraysize) then
    // bell(3)
   else
    count#:+1
    x(count#):=ix#
    y(count#):=iy#
    if count#=1 then
     t(count#):=0
    else
     t(count#):=t(count#-1)+distance(x(count#),y(
     count#),x(count#-1),y(count#-1))//wrap line
```

more»

```
    endif
    pencolor (1)
    plot ix#,iy#
   endif
  when 32 // space
   null
  otherwise
   // bell(3)
  endcase
 endwhile
 //
endproc get'points
//
proc graphics'cursor(ix#,iy#,color#)
 spritepos 1,ix#-4,iy#+2
 spritecolor 1,color#
endproc graphics'cursor
//
func distance(x1,y1,x2,y2)
 return sqr((x1-x2)^2+(y1-y2)^2)
endfunc distance
//
func same'point'again
 // TRUE if the current point is the same as
 // the most recently entered point
 if count#<1 then return false
 return (ix#=x(count#) and iy#=y(count#))
endfunc same'point'again
//
proc spline(ref f(),ref t(),count#,ref calcf(),
ref calct()) // wrap line
 // Fits a cubic spline to f(1:count#) as a
 // function of t(1:count#), then calculates
 // equally spaced values of T, places them in
 // calct(1:splinesize), computes the
 // corresponding interpolated values of F, and
 // returns them in calcf(1:splinesize).
 //
 // Adapted from the FORTRAN routine CUBSPL
 // (Carl de Boor, A Practical Guide To Splines,
 // 1978).
 //
 // Compute first difference of F and T
 for i#:=2 to count# do
  b(i#):=t(i#)-t(i#-1)
  c(i#):=(f(i#)-f(i#-1))/b(i#)
endfor i#
//
// take care of beginning of curve
c(1):=b(3)
b(1):=b(2)+b(3)
a(1):=((b(2)+2*b(1))*c(2)*b(3)+b(2)*b(2)*c(3)
)/b(1) // wrap line
//
// forward pass of  Gausian elimination
for i#:=2 to count#-1 do
 g:=-b(i#+1)/c(i#-1)
 a(i#):=g*a(i#-1)+3*(b(i#)*c(i#+1)+b(i#+1)*c(
 i#)) // wrap line
 c(i#):=g*b(i#-1)+2*(b(i#)+b(i#+1))
endfor i#
//
// take care of end of curve
g:=b(count#-1)+b(count#)
temp:=(b(count#)+g+g)*c(count#)*b(count#-1)
temp:+b(count#)+b(count#)*(f(count#-1)-f(
count#-2))/b(count#-1) // wrap line
a(count#):=temp/g
g:=-g/c(count#-1)
c(count#):=b(count#-1)
//
// complete forward pass
c(count#):=g*b(count#-1)+c(count#)
a(count#):=(g*a(count#-1)+a(count#))/c(count#)
//
// back substitution
for i#:=count#-1 to 1 step -1 do a(i#):=(a(i#)
-b(i#)*a(i#+1))/c(i#) // wrap line
//
// generate cubic coefficients
for i#:=2 to count# do
 d:=(f(i#)-f(i#-1))/b(i#)
 e:=a(i#-1)+a(i#)-2*d
 b(i#-1):=2*(d-a(i#-1)-e)/b(i#)
 c(i#-1):=(e/b(i#))*(6/b(i#))
endfor i#
//
// compute the points to be plotted as a
// function of WT
wt:=0
dt:=t(count#)/(splinesize-1)
j#:=1
```

more»

```
 for i#:=1 to splinesize do
  while t(j#+1)<wt and t(j#+1)<>0 do j#:+1
  calct(i#):=wt
  h:=wt-t(j#)
  calcf(i#):=f(j#)+h*(a(j#)+h*(b(j#)+h*c(j#)/3)/2)
  wt:+dt
 endfor i#
endproc spline
//
proc fit'curve
 plottext 1,9,spc40$
 plottext 1,1,spc40$
 hidesprite 1
 plottext 112,1,"calculating"
 spline(x,t,count#,calcx,calct)
 spline(y,t,count#,calcy,calct)
endproc fit'curve
//
proc show'curve
 pencolor (7)
 spritecolor 1,1
 for i#:=1 to count# do
  ix#:=x(i#)
  iy#:=y(i#)
  moveto ix#-4,iy#
  drawto ix#+4,iy#
  moveto ix#,iy#-2
  drawto ix#,iy#+2
 endfor i#
 moveto calcx(1),calcy(1)
 for i#:=2 to splinesize do drawto calcx(i#)
 ,calcy(i#) // wrap line
 plottext 1,1,spc40$
 plottext 64,1,"hit any key to continue"
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc show'curve
//
proc str(num,ref number$)
 number$:="   "
 repeat
  number$:=chr$(num mod 10+48)+number$
  num:=num div 10
 until num=0
endproc str ■
```

Batch File

by Doug Drake

Bat.2drive'keys is a COMAL 2.0 batch file on *Today Disk #16* which sets up the function keys for two single disk drives. My basic approach was to use the unshifted key for Unit 8 (Drive"0:") and the same function key shifted for Unit 9 (Drive"2:").

Thus to call the directory of your first drive, press F1. To delete a file from the first drive, cursor up to the directory entry and press F3. To load a file from the first drive, cursor up to the entry and press F7. For each corresponding function on the second drive, use the corresponding shifted function key. For F5/F6, I stuck in a couple of other functions I like. F5 deletes seven spaces for a **SAVE** from the program listing. And F6 is **LIST**.

Setting up the function keys is really a matter of personal preference, and a person should play around until he or she finds the setup they like best. For instance, I prefer **LOAD** to **RUN** when writing programs; others may prefer something else.

One other interesting note: with the Super Chip's **Autostart** feature, you can have your *hi* program **SELECT INPUT** a batch file to set up your function keys. Even if you don't have Super Chip, you may still want to include the function key definitions in your customized *hi* program. Here's the batchfile:

```
USE system
defkey(1,"CAT ""0:""13"")
defkey(2,"CAT ""2:""13"")
defkey(3,"DELETE"13""11""13"")
defkey(4,"DELETE"148""148""""2:"13""11""13"")
defkey(5,""20""20""20""20""20""20""20"")
defkey(6,"LIST  "13"")
defkey(7,"LOAD "13""11""13"")
defkey(8,"LOAD""2:"13""11""13"") ■
```

Calculations with Matrices

by Bill Inhelder

A set of matrix operations is not included in most computer languages because of the specialized mathematical nature of such operations. The program *matrixcalc* on *Today Disk #16* serves as a mathematical utility by providing an extensive set of such operations.

Matrixcalc presents operations which act upon a single matrix as well as a pair of matrices. The unary operations include: scalar multiplication, the inverse, the transpose, the determinant and the rank. The binary operations include: addition, subtraction and the multiplication.

From a list of matrix operations the user selects an operation and then supplies the dimension of each matrix and appropriate data. After the matrix calculation is displayed, the user may optionally select another operation which will use the displayed matrix and an additional matrix, if required by the operation. For example, the inverse of a square matrix may be selected first. That result is then multiplied by another matrix. This set of operations could be used to solve a system of linear equations. Thus to solve the following system for x ,y , z:

2x+3y+5z=3
3x+4y+7z=6
x+3y+2z=5

first find the inverse of the coefficient matrix

2 3 5
3 4 7
1 3 2

and then multiply by the column matrix

3
6
5

obtaining the solution matrix

10
1
-4

This implies that **x=10**, **y=1** and **z=-4** . Solving such a system of equations is a common problem in math and science.

Other combinations of operations are possible including an operation followed by the same operation (i.e. the inverse of the inverse). Should the calculation involve more than two operations, first find the result of two operations and copy down the result. Now run the program again and enter the result, the operation and an additional matrix if required. For example, if **A**, **B** and **C** are appropriately dimensioned matrices and **k** is a real number, the calculation **kA(B+C)** could be accomplished by:

1) **run program - enter B(,) and C(,)**
2) **add B(,)+C(,) // write down result**
3) **rerun program**
4) **enter A(,)**
5) **enter result from part 2 and multiply**
5) **multiply new result by scalar constant k**

Because of the nature of the determinant and rank functions, these operations may not be chained with any other operation. The rank function is particularly useful in determining whether a system of equations is consistent or not. If the ranks of the coefficient and augmented matrices are equal then the system is consistent.

Dimensions consistent with the various operations and combination of operations must be correct or program errors will occur. Warnings are provided throughout the program. Data entry is simplified with prompts specifying row and column entries. If you have ever had to find by hand the determinant or inverse of a 3 by 3 matrix or larger, you will appreciate the usefulness of this program. ■

Matrix Package

by Richard Bain

Several *COMAL Today* readers have sent in programs for curve fitting and programs doing matrix operations. I have wanted to write a matrix package for a long time and finally got the chance to put one together. Special thanks go to Robert Ross, whose multiplication package I borrowed. Thanks also to Bill Inhelder and Kurt Heinrich whose submissions to *COMAL Today* aided my effort to choose, implement, and demonstrate various matrix procedures. *Pkg.matrix* on *Today Disk #16* contains several matrix procedures. A short demo program on the same disk uses the package to determine a least squares polynomial fit of user data.

USE matrix // initializes package

FUNC matinverse(mat(,), REF imat(,))
Mat and imat must both be square matrices of the same dimension. Imat will normally be set to the inverse of mat. It is possible to have the same matrix passed as each parameter in the function call. This would result in the matrix being replace by its inverse. The method used for the matrix inverse is Gausian reduction with full pivoting. This method is slower, but more accurate than many other common methods. The package function is about seven times faster than the COMAL function it is based on. The result of the function is the determinant of the original matrix. If the original matrix has no inverse, then the function correctly returns zero, but the inverse matrix will be an undefined mess.

PROC matadd(REF a(,),REF b(,),REF c(,))
PROC matsubtract(REF a(,),REF b(,),REF c(,))
Mat c will be assigned the sum or difference of matrices a and b. All three matrices must be the same dimension.

PROC matfill(x, REF m(,))
Each element of m is set to the value of x.

PROC matid(REF m(,))
M must be a square matrix. It is set to the identity matrix. For example, **m(i,i):=1**, but **m(i,j):=0** if **i<>j**.

PROC matscale(x, REF m(,))
Each element of m is multiplied by x.

PROC matcopy(REF a(,), REF b(,))
Mat b is set to the value mat a. Both matrices must contain the same number of elements, but don't need to be exactly the same dimension. For example, a 3 by 3 matrix could be copied to a 9 by 1 column matrix.

PROC mattranspose(a(,), REF atran(,))
Atran is assigned the transpose of a. For example, **atran(i,j):=a(j,i)**. If a is an m by n matrix, then atran must be an n by m matrix.

PROC matmult(a(,),b(,),REF c(,))
C is assigned the product of a and b. This is the procedure by Robert Ross from *COMAL Today #15* with two changes. The first change is that I used an unreleased version of his procedure which doesn't do the internal rounding necessary to let the procedure exactly match the results of the equivalent COMAL code. This version is faster and I believe more accurate than the earlier version. The second change was to use value parameters for matrices a and b. This allows matrix c to obtain the correct product in the special case where the same matrix is passed to c and either a or b. Note: programs written to use the old procedure can use this one instead without any other changes.

FUNC version'matrix$
This function returns the current version of the Matrix package.

Further reference:

Matrix Use, COMAL Today #15, page 77 ■

Fast Fourier

by Bill Inhelder

The article *FFT* by Tom Kuiper, in *COMAL Today #11* suggesting a fast Fourier transform program making use of filtering, brought to mind a program I wrote in 1981. It made use of an algorithm by P.L. Emerson published in *Creative Computing, July 1980. Fast'fourier* on *Today Disk #16* was written in COMAL 2.0 and incorporates filtering, graphics and screen dumps. It requires that the values of a user-determined waveform be entered. The program then produces a series of equations which approximate that waveform. The graph is produced from the equations and is superimposed on the graph of the user's set of data points. Various components (harmonics) can then be filtered out observing the effect on the remaining waveform by graphing it.

The final graphics screen may be dumped to a printer by using Randy Finch's *pkg.finchutil* described in *COMAL Today #10*. Lines 310 and 600 can be changed to accommodate printers other than the *Star Gemini-10X* for which this program is intended.

Program Description

This program determines the equations of the harmonic components of any user-defined waveform and graphs any subset of the harmonic equations of that waveform. Although a knowledge of high school trigonometry is useful to understand the nature of the equations and the ultimate effect on the waveform, anyone who can draw a waveform on a sheet of graph paper and determine the y-value of the data points can use the program and derive some meaning from it.

The following input conditions must be met:

1. Enter the number of points, *N*, in the user defined waveform. It must be a power of 2 less than or equal 64 (ie. 8,16,32,64).
2. Enter the y-values of the points one at a time. The *N* points **must** be equally spaced on the x-axis.
3. Type the letter **c** to continue after each graph is produced and follow the instructions on the textscreen.

The program produces the following output:

1. A graph of the input points of the waveform scaled to the full size of the screen together with the axes.
2. A table of equations, called spectral components, for the d.c. and harmonic components of the waveform. The harmonic components appear in pairs in the following form:

d.c. component	y(0)=a0cos(2pi t/T)
1st harmonic	y(1)=a1cos(2pi t/T) y(1)=b1sin(2pi t/T)
2nd harmonic	y(2)=a2cos(4pi t/T) y(2)=b2sin(4pi t/T)
nth harmonic	y(n)=ancos(2npi t/T) y(n)=bnsin(2npi t/T)

   where 0<=t<=T and T is the period of the waveform. In this program T will be considered as 1 unit of time (i.e. 1 sec, 1/250 sec, 1/1000 sec). Note that the number of pairs of harmonic components is equal to half the number of points entered.

3. An optional hardcopy of the table of equations may be obtained.
4. Using the d.c. component and harmonic components a graph of the waveform is optionally produced.
5. The screen display of the graph produced by the equations overlaid on the original graphics display may be optionally dumped to the printer.

more»

6. Any specified harmonic pair(s) can be removed (filtered) from the table of equations and the graph of the remaining components produced. This process can be repeated with other specified pair(s) resulting in the cumulative removal of harmonic pairs together with an accompanying graph of the remaining components. At each stage the screen display may be optionally dumped to the printer.
7. To exit press **n** in response to any further filtering.

On a sheet of graph paper draw a waveform of 16, 32, or 64 points in length. *Note 15 squares produce 16 points*. The beginning and ending points should be the same because this will represent one cycle of the wave. Record the y-values for each of the points of the waveform. You now have the 16, 32 or 64 points to enter in the program. The points of the waveform may be all above the x-axis or above and below, but not all below the x-axis. All sorts of special waveforms are possible: square waves, saw-toothed waves, d.c. wave (a line), damped waveform, or a waveform representing the sounding of a vowel.

Sixteen data points produce a good approximation of the original waveform. 32 data points require about 2 1/2 minutes computing time for the Fourier equivalent waveform, producing a better approximation. Sixty-four data points produce an excellent fit to the original waveform.

To produce the vowel sound of the letter "a" spoken at a pitch of 260 cps (T=1/260 sec.) would require that oscillators create and mix the signals of sine and cosine waves of the appropriate frequencies and amplitudes.

As an example, one such waveform might have the following 16 y-values:

2, .8, -2.4, -3.2, -2.3, -4.25, -1.2, 3, 6, 3.8, -2, .8, 2.8, -1, -3.5, 2

Now **RUN "fast'fourier"** and type in **16** for the number of points. Next type in each of the y-values on a separate line and press *«return»*. Follow the instructions of the program. Remember to press **C** to continue after each graph is drawn. Be sure the printer is turned on if a screen dump is desired.

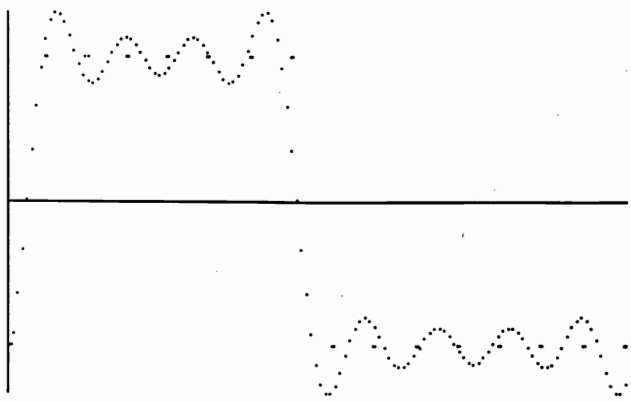
The filtering of the harmonic pairs is cumulative, which means that the program is not designed to restore any pair once it has been filtered out.

We can achieve a better approximation to the first waveform example using 32 y-values:

2, 1.8, .8, -1, -2.4, -3.6, -3.2, -2.3, -2.3, -3.6, -4.25, -3.3, -1.2, .9, 3, 5, 6, 5.5, 3.8, -.4, -2, -1.5, .8, 2.8, 2.8, 1, -1, -3, -3.5, -2, 1, 2

Try a square wave:

-1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1



or a saw-toothed wave:

0, 1.1, 2.2, 3.3, 3.75, 2.83, 1.6, .5, -.5, -1.6, -2.83, -3.75, -3.3, -2.2, -1.1, 0

Try some of your own. Even though it is more time consuming, more points produce a better approximation. ■

Stacks Revisited

by Richard Bain

[In COMAL Today #14 I wrote an introduction to COMAL 2.0 stacks. I received several requests for more details. This is what I think is placed on the Commodore COMAL 2.0 stacks. This article is not for beginners.]

Important Zero Page Pointers (2 bytes each):

$0e LOCLPT - upper stack chain
$10 FORPT - lower stack chain
$16 SPROG - start of program
$18 SVARS - start of variable table
$1a SSTACK - start of lower stack
$1c SMAX - start of upper stack
$2d STOS - end of lower stack
$2f SFREE - end of upper stack
$34 SCLSD1 - current scope - upper stack
$36 SCLSD2 - current scope - lower stack

Data Structures:

Three kinds of information are stacked: intermediate calculations, values of variables, and system information such as what line to return to when a procedure ends. Intermediate calculations are temporary (they don't remain on the stack during execution of more than one program line) and will not be discussed here. Variable storage will be discussed in detail now, and system information will be introduced.

To understand the *value* of a variable (beyond the value of a is 1), one must first understand what *types* of variables exist and then what *bytes* get placed on the stack for each *type*. For now, I am defining variable to include any COMAL entity which is given a name table entry. The name table is organized as follows:

length	type	address	name

Length, one byte, is the length of the entire entry (4 more than the number of characters in the name). Type, one byte, will be discussed below. Address, 2 bytes - (lo/hi), is the absolute starting address of where the *value* of the variable is stored. Name is the Commodore ASCII representation of the variable's name.

Name table entries start at the address pointed to by SVARS and continue one after another until the end of the table. The end of the table is denoted by the *last* name table entry having a zero in the length byte.

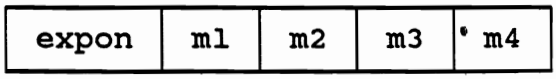
The *type* can best be understood by analyzing the type byte from the name table entry. It has eight bits (bit 7 is the most significant bit and bit 0 is the least significant bit):

bit 7:	1 = array, 0 = normal
bits 6,5:	00 = normal
	01 = value parameter
	10 = reference parameter
	11 = not used
bit 4:	1 = variable has a value
	0 = variable has not been used
bit 3:	1 = package name, 0 = normal
bits 2,1,0:	000 = real variable
	001 = integer# variable
	010 = string$ variable
	011 = label:
	100 = procedure
	101 = real function
	110 = integer# function
	111 = string$ function

These bits are **ORed** together to form the variable type byte. For example: %00010000 represents a real variable, %01010110 represents an imported integer# function.

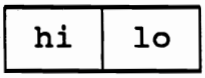
Now let's examine what is placed on the stack for each of these different types:

more»



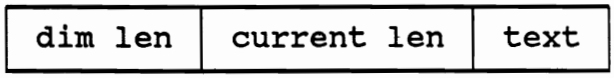
Real variable: 5 bytes. Its value can best be shown by this program fragment:

```
IF expon=0 THEN
  value:=0
ELSE
  sign:=(128>m1)*2-1
  IF sign=1 THEN m1:+128
  power:=2^(expon-160)
  mantissa:=m4+256*(m3+256*(m2+256*m1))
  value:=sign*power*mantissa
ENDIF
```

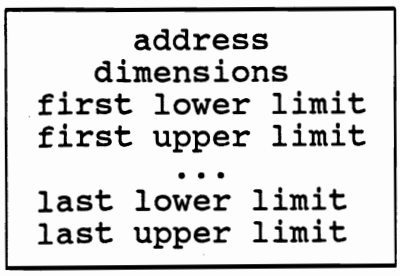


Integer# variable: 2 bytes in (hi/lo) format.

```
value:=256*hi+lo
IF value>=32768 THEN value:-65536
```



String$ variable: 4 bytes plus dimensioned length. First two bytes are the dimensioned length in (hi/lo) format, next two bytes are current length (hi/lo). The rest of the bytes are the Commodore ASCII representation of the string's characters.

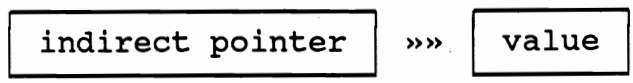


Array variable: value is called a dope vector. First two bytes are the address (lo/hi) of the first element in the array (real, string$, or integer#). The next byte contains the number of dimensions in the array. There are an additional 4 bytes for each dimension - 2 bytes each (hi/lo) for the lower and upper limits. The elements of the array are stacked consecutively by rows.

Label: 2 bytes - the address (lo/hi) of the start of the COMAL program line containing the label.

Procedures and functions: 3 bytes - First two bytes are the address (lo/hi) of the COMAL program line containing the procedure or function header. The next byte is the page it is on, normally 0. If the procedure or function is from a package instead of the program text, the address is 3 bytes before its header and the page is the page of the package.

Call by value parameters: treated like normal real, string$, or integer# variables.



Call by reference parameters and imported variables: 2 bytes - address (lo/hi) where the variable is stored in the manner described above.

Run-Time Stack Entries:

When a program is **RUN**, all variables are cleared. Among other things, this involves placing a 0 in the type bytes of their name table entries. SFREE and STOS are set to the values of SMAX and SSTACK to clear the stacks. The high bytes of LOCLPT and FORPT will be set to 0 indicating there are no stack chains, and the high bytes of SCLSD1 and SCLSD2 will be set to 0 indicating that everything is in the current scope until a **CLOSED** procedure or function is entered. The scanning process will then put the values of the procedures and functions on the

more»

upper stack and adjust SFREE accordingly. In general, when a variable is encountered for the first time, its value will be placed on the upper stack. More on this next issue.

Sample Stack Entry:

When a COMAL structure such as a **FOR** loop is entered, COMAL creates a stack entry to maintain system variables. The first 2 bytes to be stacked (at the location pointed to by STOS) are the address bytes pointing to the previous stack entry. This address is found in FORPT. Now FORPT can be given the value from STOS to point it to the new stack entry. STOS must also be updated to continue pointing to the top of the stack. The next byte to be stacked tells what kind of structure was just entered. In the example of the **FOR** loop with a real control variable, this byte is a 0. Note: stack entries for other structures will be discussed next issue.

The control variable, say x, in COMAL 2.0 is local. This means that whatever value x had before the **FOR** loop is entered must be restored when the **FOR** loop ends. One might expect COMAL to place the old value on the stack and put the local value where the old one was. This simply won't work! What if x used to be a string? Therefore, COMAL puts the old name table entry of x on the stack. It creates a new place for the new value, and leaves the old value alone. Let's examine the specific bytes involved.

The fourth and fifth bytes of the **FOR** loop stack entry contain the address of the control variable's name table entry. (There is only one name table entry per variable - it doesn't move just because the identity of the variable is being changed.) Then 3 bytes (the type byte and 2 address bytes) from the old name table entry are stacked (the length of the name table entry and the name of the variable don't need to be stacked). The next 5 bytes to be stacked are for the value of the control variable, initialized to its starting value. The address bytes of the name table entry will be set to point here; the type byte will also be set to $10 denoting a real variable. Ten more bytes are then stacked (bytes 14 - 23 in the stack entry). The first 5 contain the number the **FOR** loop counts to; the last 5 contain the **STEP** value (usually 1). The stack entry is now complete and 23 is added to STOS.

2 bytes	old FORPT
1 byte	0 - real FOR loop
2 bytes	name table entry addr
3 bytes	old name table entry
5 bytes	value of counter
5 bytes	ENDFOR value
5 bytes	STEP value

When COMAL reaches **ENDFOR** for the last time, it is very easy to restore the stack to what it was before the **FOR** loop was entered. (Next issue we will see how the **FOR** loop stack entry might be buried by other stack entries. However, by the time COMAL reaches **ENDFOR**, the other stack entries will have been removed. Thus the original **FOR** loop stack entry will always be on top of the stack when COMAL reaches **ENDFOR**.) FORPT still points to the start of our stack entry. We know the stack entry offsets for the address telling where to put the old name table entry of x. We know to place the value of FORPT back to STOS and to place the stacked value of FORPT back there too. Now the stack looks exactly like it did before the **FOR** loop was ever called. This makes it possible to do things like having nested **FOR** loops, both using control variables with the same name. The creative and devious could even use this information to place the old name table entry of x back into some other variable such as y when the **FOR** loop ends. Neat huh.

[For those interested in learning more, I strongly recommend the using the monitor from the cmon package on Today Disk #10 to examine and change memory from a running program.] ■

Apple COMAL

COMAL is an ideal first programming language. It was originally designed as a language taught in schools in place of BASIC and Pascal. Unfortunately, Apple computers are most common in our schools here, yet an implementation of COMAL for the Apple is not available. I'd like to rectify that situation this year.

My goal is to produce a version of COMAL to run on the 128K Apple IIe and IIc that is compatible with the COMAL kernal. It will provide full screen editing, easy device access, and at least 32K of user program space. It will run under PRODOS which provides subdirectories.

Since I can't afford a complete Apple computer system, I am developing Apple COMAL in several stages. Currently I am utilizing equipment that I already own or have access to.

First I will develop version 1.0 which would have most of the capabilities of C64 COMAL 0.14, plus many added commands that 0.14 lacks, such as **page, cursor, print at, val, str$,** and **get$.**

After that, version 2.0 will be developed. It will add most of the new features available in other COMAL 2.0 implementations.

This project has been under development for some time, and is reaching the point where the first preliminary version of Apple COMAL should be ready within a few months.

I am developing Apple COMAL without any financial backing in order to keep control of its implementation and marketing. In order to generate the capital necessary to complete this project, I am publishing a newsletter on its development. *Apple COMAL Notes* will do more than show the current state and capabilities of Apple COMAL; it will give technical information on how COMAL works and ask questions of its readers on how to implement certain features. For example, what should **"" IN "abc"** return?

This is your chance to see and influence how COMAL is developed on the Apple II (although the standard KERNAL will still be followed to maintain compatibility).

I realize that few *COMAL Today* readers own an Apple computer. I hope that you are interested enough in seeing COMAL developed in the United States that you will find people who own Apple computers and convince them to invest in COMAL's future. Please spread the word to your friends and schools. Help make it possible for your children to learn COMAL in schools with Apple computers.

A 10 issue subscription to *Apple COMAL Notes* costs $25. Advanced COMALites should be interested in following the development of Apple COMAL even if they don't own the computer.

For $50 you will receive *Apple COMAL Notes* until Apple COMAL version 1.0 is available for sale, when you will be sent a disk copy.

For $75 you will receive version 1.0, when available, and *Apple COMAL Notes* until Apple COMAL version 2.0 is available, when you will be sent a disk copy.

For $100 you will receive *Apple COMAL Notes* for as long as it is published, and all disk upgrades free of charge.

I do not believe in copy protection, so you will be free to make backups of your disks for your own use only, but not to give to other people. I ask that you think about what it costs me to develop this language, both in time and money.

Please write for further information. Checks should be made payable to:

David Stidolph, COMALites United
1670 Simpson #102, Madison, WI 53713
(608) 222-4505 - Evenings only ■

Today Disk #15 - Front

105 Files | 5 Blocks Free:

boot c64 comal	wheel'of'fortune	create.proc	print'at.proc	sound'system.lst
c64 comal 0.14	----------	cursor.proc	quicksort.proc	----------
comalerrors	- functions -	dir.proc	repeatkeys.proc	- data files -
ml.sizzle	----------	drive8.proc	save'obj.proc	----------
hi	curcol.func	drive9.proc	saveshape.proc	- do not load -
menu	currow.func	expand'ram.proc	settime.proc	----------
window-rs232.obj	enhanced.func	fillkeys.proc	shift'wait.proc	hrg.wheel
----------	file'exists.func	getbackgrnd.proc	str.proc	load/save.mem
-comal programs-	free.func	inkey.proc	textcolors.proc	ran.doctorwho
----------	freefile.func	input'at.proc	turbo.proc	ran.wheel
1520-3d'airplane	gettime.func	joystick.proc	----------	----------
buffer'display	pi.func	koala.proc	- listed files -	-for more info -
doctorwhodb	round.func	lightpen.proc	----------	- send sase to -
draw'universe	trunc.func	load'errors.proc	-several procs -	----------
edit'wheel'data	val.func	load'obj.proc	-and funcs that-	- comal users -
kaprekar	----------	loadfont.proc	-work together -	- group, usa -
merge'procs	- procedures -	loadshape.proc	----------	- 6041 monona -
print'captured	----------	mount.proc	1520driver.lst	- madison, wi -
print'dr'who	call.proc	paddle.proc	1571.lst	- 53713 -
qlink'to'listing	circle.proc	page.proc	bit'funtions.lst	-(608)222-4432 -
recursivedesigns	clearkeys.proc	plot'text.proc	read'block.lst	----------

contributors:

Jim Adams

Richard Bain

Jack Baldrige

Captain COMAL

Herbert Denaci

Len Lindsay

D Bruce Powell

David Stidolph

plus many more contributed the procedures and functions

Today Disk #15 - Back

83 Files | 2 Blocks Free:

hi	demo/tron	----------	func.mid2$	----------
----------	demo/turnto	bf.birds of prey	func.right$	- listed file -
-comal programs-	doctorwhodb	bf.chickadee/tit	----------	----------
----------	draw'snowflake	bf.jays/crows	- procedures -	lst.turnto
bird'data'corr	draw'universe	bf.woodpeckers	----------	----------
bird'filer	kaprekar	dat.bird'master	proc.type	-for more info -
buffer'display	list'dr'who	dat.capitols	----------	- send sase to -
can'you'tell'me	print'captured	dat.quiz'menu	- packages -	----------
chip.airplane	print'dr'who	dat.rotate1	----------	- comal users -
demo/echo	qlink'to'listing	dat.rotate2	pkg.echo	- group, usa -
demo/irq1	recursivedesigns	dat.rotate3	pkg.irq	- 6041 monona -
demo/irq2	rotate	dat.world'cap	pkg.matrix	- madison, wi -
demo/irq3	sky'view	----------	pkg.text	- 53713 -
demo/irq4	tempered'chords	- functions -	pkg.windows	-(608)222-4432 -
demo/matrix	type'quick	----------	src.irq	----------
demo/statistics	----------	func.left$	src.matrix	
demo/text	- data files -	func.mid$	src.text-1.02	

contributors:

Jim Adams
Richard Bain
Norbert Bakker
Jack Baldrige
Captain COMAL
Herbert Denaci
Bob Hoerter
Bill Inhelder
Dick Klingens
Len Lindsay
D Bruce Powell
Joel E Rea
Robert Ross
Richard Shagott
Todd Shagott
David Warman
Lowell Zabel

User Group Disk #14

60 Files | 11 Blocks Free:

hi	day'of'the'week	text'to'hires	dat.family'file	-freely copied -
----------	dual'moire	weight&balance	dat.journal	----------
-comal programs-	home'finance	----------	nova68	-for more info -
----------	journal'account	- text files -	test'param	- send sase to -
-comal 2.0 only-	point'of'sale	----------	test.ran	----------
----------	rnd'file'create	txt.home finance	test.scr	- comal users -
1520-alfred'e	rnd'file'update	txt.w&b	tth.ml	- group, usa -
auto'database	screen'construct	----------	----------	- 6041 monona -
bridge'evaluator	show'holidays	- data files -	-this disk may -	- madison, wi -
char'by'char	sort'and'report	----------	-be included in-	- 53716 -
check'files	spreadsheet	dat.assets 2	- user group -	-(608)222-4432 -
christmas'carol	ten'key	dat.assets 3	-libraries, and-	----------

contributors:

Jack Baldrige
Eric Haas
Brian Hogue
Bill Inhelder
Alex Jackson
Dick Klingens
Si LaBar
Bill Nissley
David Lee Powell
G R Roberts

Index Disk - Front

38 Files | 229 Blocks Free:

----------	idx.print	- programs -	authors	- group, usa -
- comal 2.0 -	idx.x-reference	----------	keywords	- 6041 monona -
- programs -	idx.edit	see'index(0.14)	titles	- madison, wi -
----------	idx.sort	----------	----------	- 53716 -
see'index(2.0)	idx.count	- index for -	-for more info -	-(608)222-4432 -
idx.enterdata	idx.sort-all	- comal today -	- send sase to -	----------
idx.parse/split	----------	- issues 1-12 -	----------	
idx.sort/cite	- comal 0.14 -	----------	- comal users -	

contributor:

Kevin Quiggle

This is the matching disk to the INDEX book. It has the author, title, and keyword data files on it, plus a reader program for both COMAL 0.14 and COMAL 2.0.

Index Disk - Back

20 Files | 172 Blocks Free:

----------	keywords.c-d	keywords.s-z	----------	- madison, wi -
-keyword files -	keywords.e-g	----------	- comal users -	- 53716 -
----------	keywords.h-n	-for more info -	- group, usa -	-(608)222-4432 -
keywords.a-b	keywords.o-r	- send sase to -	- 6041 monona -	----------

Order Form (MORE ITEMS ON OTHER SIDE)

Name:_____ SUBSCRIBER NUMBER:_____ Feb 1987
(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

[X] 24.95 **COMAL Today newsletter-> How many? 10 Start at: [_] renew [X] 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/$2.95 COMAL Today backissue: circle #s wanted-> 5 6 7 8 9 10 11 12 13 14 15 16
[ ] _____ $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 16**
(Disk subscriptions must be 6 or more disks - single Today Disks are available - use above line)

SYSTEMS:

[ ] _____ $138.95/$128.90 Deluxe Cartridge Pak (black C64 COMAL 2.0 cartridge)-(shipping add $5)
(New Pak also includes: two manuals, 4 disks, and Super Chip)
[ ] _____ **$88.95/$85.95 Black c64 COMAL 2.0 Cartridge, Plain (no books/no disks)-(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.**
[ ] _____ **$29.95/27.95 COMAL 0.14 Starter Kit (4 disks, 2 books, 6 newsletters, more)-(ship add $4)**
(includes COMAL From A to Z book, Workbook, Demo, Tutorial, and Best Of disks)
[ ] _____ $595/$595 UniComal IBM PC COMAL 2.0 (English manual/program disk)-(shipping add $5)
(system is in English ... for 256K IBM PC Compatibles ... prepaid only - special order)
[ ] _____ **under $100 ... CP/M COMAL 2.0 expected soon! Imported from Denmark**
[ ] _____ **under $100 ... Mytech IBM PC COMAL 2.0 expected soon! Imported from Sweden**
[ ] _____ **under $100 ... Apple IIe / IIc COMAL ... end of 1987. Made in the USA.**

DISKS:

[ ] _____ **$14.95/$9.75 Math & Science Disk (COMAL 2.0)**
[ ] _____ $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 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 14**
[ ] _____ $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 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!)

Total_____ + _____ Shipping (this side) (Canada & First Class add $1 extra per book)
Total_____ + _____ Shipping (other side)
(overseas shipping is extra)
Total 40.85 + 4.00 = US$ 44.85 **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

Order Form (MORE ITEMS ON OTHER SIDE)

Name:_____ SUBSCRIBER NUMBER: 173 Feb 1987
(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)

[ ] _____ **$15.95/$13.95 Graph Paper - book / disk set by Garrett Hughes - (shipping $2)**
(Due to be released in April 1987. It is a wonderful system for function graphing)

[X] 4.95 **$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!)

[X] 4.00 **with your INDEX get as many COMAL Today backissues as you need - only 50 cents each**
(Circle ones wanted: 5 6 7 8 9 10 11 12) - free UPS ship--1st Class/Canada $1 each

[ ] _____ **$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)

[ ] _____ **$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) (with free white case)

[X] 4.95 **$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 40.85 + 4 Shipping = US$ 44.85 **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

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.