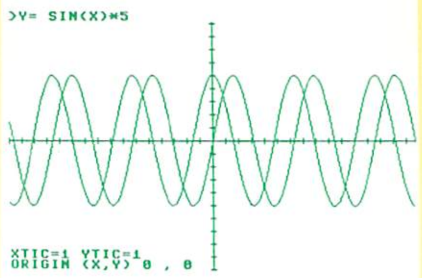
COMAL TODAY 14


```
>Y= SIN(X)*5

XTIC=1 YTIC=1
ORIGIN (X,Y) 0 , 0
```

Multi-function Graphing
page 24

PROGRAM CHALLENGE page 59

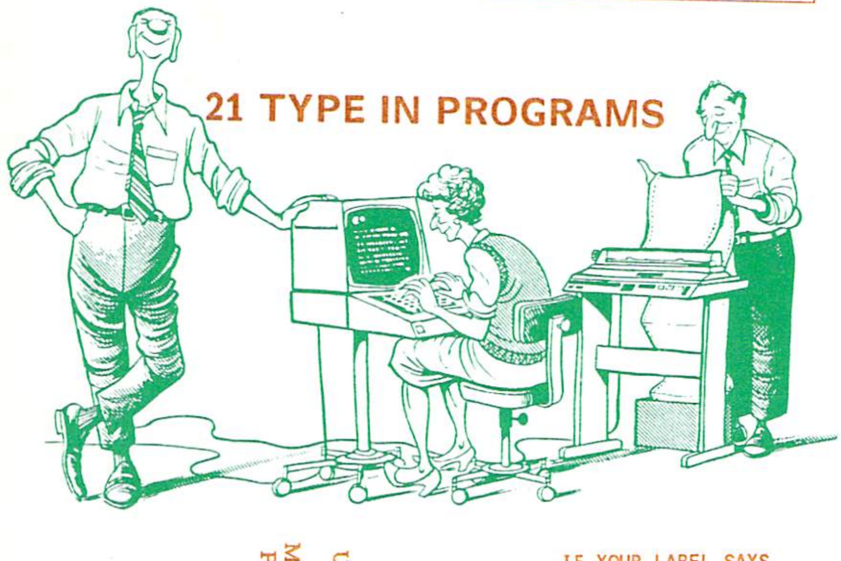
OCTOBER 1986

SUN	MON	TUE	WED	THU	FRI	SAT
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

CALENDAR page 34

COMAL Today
6041 Monona Drive
Madison, WI 53716

21 TYPE IN PROGRAMS



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

IF YOUR LABEL SAYS
LAST ISSUE: 14
YOU MUST RENEW NOW.
USE ORDER FORM INSIDE

File Name Conventions

We try to include more than just COMAL programs on our disks. We also include linkable packages, custom font definition files, hi-res bitmaps, and more. Each type of special file has been assigned a special filename suffix or prefix. Whenever possible we use these conventions in naming files for our disks. This way, just from the name of the file you can determine what type of file it is. You can tell the difference between a procedure and a package stored on disk. The package file will begin with pkg.. A COMAL 2.0 procedure filename will begin with proc. and a COMAL 0.14 procedure filename will end with .proc. Please use these naming conventions on disks that you send to us.

Here are our FILENAME CONVENTIONS:

Suffixed	Prefixed	*Meaning*
NAME	NAME	*COMAL program file*
NAME.L	LST.NAME	*Program listed to disk*
	DSP.NAME	*Prog displayed to disk*
NAME.PROC	PROC.NAME	*PROC listed to disk*
NAME.FUNC	FUNC.NAME	*FUNC listed to disk*
NAME.DAT	DAT.NAME	*Data file*
NAME.TXT	TXT.NAME	*Text file*
NAME.DOC	DOC.NAME	*Documentation file*
	EXT.NAME	*External PROC/FUNC*
	SHAP.NAME	*Sprite shape file*
	FONT.NAME	*COMAL font file*
	FONT.MC.NAME	*Multicolor font file*
	SET.NAME	*BASIC type font file*
	PKG.NAME	*Package file*
	BAT.NAME	*Batch file*
	SNG.NAME	*Song file*
	HRG.NAME	*Color COMAL picture*
NAME.HRG		*Black/White bitmap*
	CRG.NAME	*Compacted color pix*
NAME.CRG		*Compacted B/W bitmap*
	ICON.NAME	*Print Shop type Icon*
	SCRN.NAME	*Text Screen File*
	POP.NAME	*Mergeable Popover*
NAME.POP		*Program with Popover*
	CHIP.NAME	*Program for SuperChip*
NAME.BASIC		*A BASIC program*
	MEM.NAME	*A memory image file*
	LIB.NAME	*Assembly library file*
	SYM.NAME	*Assembly library file*
	SRC.NAME	*Assembly source code*

WCCA PRESENTS

THE COMMODORE SHOW

SEPTEMBER 20 & 21 1986
SHOW TIMES 10AM — 6PM
LOS ANGELES AIRPORT HILTON
CALL 213-410-4000 for hotel reservations

- *EXHIBITS, EVENTS, AND DOOR PRIZES*
- *NATIONAL COMMODORE SPEAKERS*
- *SHOW SPECIALS & DISCOUNTS*
- *SEE THE LATEST INNOVATIONS IN HARDWARE/SOFTWARE TECHNOLOGY FOR THE COMMODORE MARKET*

The only West Coast exhibition and conference focusing exclusively on the AMIGA Commodore 128 PC and C-65 marketplace

REGISTRATION FEES: ONE DAY $10.00 TWO DAYS $15.00

FOR MORE INFORMATION AND DETAILS CONTACT:
WEST COAST COMMODORE ASSOCIATION, INC.
P.O. BOX 210638
SAN FRANCISCO, CALIFORNIA 94121
(415)982-1040 BETWEEN 8AM-5PM PST

Ask Someone Who Knows

If you enjoy **Jim Strasma's** many books, and his articles in this and other magazines, you'll be glad he also edits his own highly-acclaimed computer magazine, now in its sixth year of continuous publication. Written just for owners of Commodore's many computers, each **Midnite Software Gazette** contains hundreds of brief, honest reviews.

Midnite also features timely Commodore news, hints and articles, all organized for instant reference, and never a wasted word. Whether you are just beginning or a long-time hobbyist, each issue will help you and your computer to work together effectively.

A six issue annual subscription is $23. To subscribe, or request a sample issue, just write:

MIDNITE SOFTWARE GAZETTE
P.O. Box 1747
Champaign, IL 61820

You'll be glad you did!

COMAL Today Issue 14 September 2, 1986

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
Richard Aurland
Phil Bacon
Phyrne Bacon
Richard Bain
Robert Bain
Jack Baldridge
C Barringer
Marcel Bokhorst
Michele Bostdorff
Will Bow

CONTRIBUTORS
Lewis Brown
Reed Brown
Ray Carter
R Clark
Marc Clifford
Captain COMAL
John Eldredge
Lew Fleishman
Galen Foreman
Arni Geirsson
Brian Grainger
Tony Granata
Chris Green
Eric Haas
Craig Hardy
Gerald Hobart
Bob Hoerter
Bill Howard
Bill Inhelder
Jim Kaminski
John Kleczewski
Dick Klingens
Johnny Laprarie
L LaBar
Len Lindsay

CONTRIBUTORS
David Martin
Ed Matthews
Ted Mayes
Bob McCauley
Grant McConnell
Kevin Page
C Peabody
Joe Postma
David Powell
Kevin Quiggle
Mike Rose
Robert Ross
Patrick Roye
Bernhardt Sandler
Robert Shingledecker
Stacy Sominski
William Staneski
Fred Staudaher
David Stidolph
James Synnamon
Lowell Toms
David Warman
Larry Winkles
Tony Wright
L Zabel

From the Editor's Disk

by Len Lindsay

Using small icon pictures to identify what version of COMAL applies to each article seems to be working well. The **disk** means 0.14, the **cartridge** means 2.0, the **I** means IBM PC COMAL, and the **chip** means 2.0 with Super Chip.

Installing hundreds of Super Chips lead to a very interesting discovery. We found three different sets of ROMs. We checked, and it seems the code in all three sets is identical. What is interesting is that three different chip companies have produced COMAL ROMs. To produce a ROM you usually need to order at least 10,000 of them. Now, if Commodore has already gone through three sets, that must mean that COMAL is doing quite well - at least in Europe. For the curious, here are the markings on the ROMs we found:

AMI 8550 PDB	and	AMI 8550 NBA
MOS 9256C-0051	and	MOS 9256C-0050
GI 9256C-0049	and	GI 9256C-0048

It seems that half our readers want lots of program listings, while the other half would prefer less listings. They get the *Today Disks* and can see any listing they wish. This issue has 21 programs listed. And the articles are quite good as well.

Which articles did you like the best? Which didn't appeal to you? Which program was your favorite? Do you want more technical articles? More step by step instructions? More sample screen dumps and program printouts?

You can help us determine how to fill up the 80 pages in each issue. Fill out the card inserted in this issue and mail it to us. Use another sheet of paper for your comments, if any. This will give us an idea of what you are looking for in *COMAL Today*.

I warned you last issue. Now it happened. The subscription rate has gone up. This is the first increase since *COMAL Today* began 3 years ago. That rate was based on 28 page issues without a nice cover. Since then, the size has trippled and includes a heavy printed cover. The rate increase was minimal, but if you subscribe or renew before the next issue comes out, we will give you the old rate (*save $4*).

New COMAL books? Holy smokes! We have several new books that are **almost** ready, including the index to *COMAL Today*. We are still trying to recover from the Super Chip project! Delays in editing and preparing the new books resulted. Meanwhile, the CEOS project mentioned last issue has been put on hold until we get caught up.

Mark Evans has been quite busy lately as well. He is lucky to have a test version of MacIntosh COMAL. We hoped to have an article about it for this issue, but it had to be delayed. If he gets the time to prepare the article for us, we plan to print it.

West Coast COMALites: make sure you come to the L.A. Commodore Show September 20 and 21 (their ad is on the inside front cover). I'll be there to give a one hour talk on COMAL. The rest of the show I probably will be nearby the Transactor booth. See you there.

Oh, yes. I just have to point out to you that we really cleaned up on this issue - it includes both SCOPE and LISTERINE (pages 22 and 60). And check out the program challenge on page 59. We already have a similar puzzle program from Dick Klingens called *Rotate* that we are saving to publish with the winning entry to the puzzle challenge. ▣

COMALites Unite!

by David Stidolph

COMAL 0.14

In the past it has been necessary to use the command **open file 255,"",unit 4,7** before a **select output "lp:"** in order to get the printer to print in lower case. The following poke will set the default secondary address to 7 (lower case) instead of 0 (upper case).

POKE 26131,7

One of the differences between COMAL 0.14 and 2.0 is that 0.14 lists programs to disk without indentation, and 2.0 lists with indentation. The following two pokes will make COMAL 0.14 list to disk with indentation. This is good news for people wishing to list programs to disk to print in magazines and newsletters. There is one intereasting side effect to this though. The EDIT command will now list with indentation at the beginning of the line (although it will not insert spaces on wrap lines). The two pokes are:

POKE 26521,234
POKE 26522,234

Many of our best COMAL programmers that submit their software for publication have gone on to COMAL 2.0 (this does make some sense). Unfortunately this means that we are recieving less 0.14 material. Thanks to the compatiblity of COMAL we can still convert many of the shorter 2.0 programs to 0.14. Please consider sharing your programs with other struggling programmers. We will send you a User Group disk of your choice in exchange. The programs we publish reflect what other users of COMAL wanted written and went out of their way to share.

COMAL 2.0 NEWS

Last month I released a C128 package. After the disk was mailed we found that certain registers in the 80 column video chip produced different results for different monitors. After some experimenting and running all over town to check different model C128's we settled on numbers which worked on all monitors. On *Today Disk #14* is a new C128 package that will initialize the 80 column screen properly. Super Chip was made after being updated with these numbers.

It's been over one month since the release of Super Chip. We worked hard to make Super Chip error free, but with over 16,000 bytes of machine code it is hard to be sure. See the Super Chip update article on page 48 for one quirk.

I just got back from the Fox Valley Computerfest, and I was pleasently surprised by the number of programmers intereasted in learning COMAL. Normally few programmers go to conventions.

NEW MODEM DISK

Lately we have received several new modem programs, procedures and functions. All of these and a public domain **x-modem** program have been added to the modem disk. Please see the modem disk catalog this issue.

NEW HI PROGRAMS

Today Disk #14 features new HI programs for both sides of the disk. The 0.14 program still loads error messages and expands memory, but is erased with a NEW command so you start fresh. The 3 POKE's printed above are also included. The 2.0 HI program was written by Eric Hass and is a "must see." ■

Printer and Plotter Notes

LOWER CASE TO PRINTER - 0.14

File number 255 is the printer file. COMAL 0.14 opens the printer by default in upper case mode. Override this by opening the printer yourself before selecting it:

```
open file 255,"",unit 4,7,write
select "lp:"
```

You also can overwrite the default printer secondary address so that it normally will produce lower case:

```
poke 26131,7
```

OTHER DEVICE PRINTER - 0.14

COMAL 0.14 uses device 4 as the printer device. If your printer is assigned another device number, such as 5, you can override COMAL's default by selecting the printer yourself first:

```
open file 255,"",unit 5,7,write
select "lp:"
```

Note: the 5 is the device number and the 7 is the secondary address.

1520 PLOTTER - 0.14

If you wish to use your plotter as the printer from COMAL 0.14, you can open the printer file yourself before each select:

```
open file 255,"",unit 6,0,write
select "lp:"
```

Note: the 6 is the device number for the plotter and the 0 is the secondary address. You can also change the default device number to 6 so that the plotter is always selected:

```
poke 26129,6
```

CHANGE PRINTER DEFAULT - 2.0

COMAL 2.0 has one command to set all the printer defaults. Setprinter is in the system package and has several optional parameters. The example uses the defaults:

```
USE system
setprinter("u4:/a-/l-/t+/s7/d-")
SELECT "lp:"
```

u followed by device number and colon. Change 4 to 6 to print to a plotter.
a followed by a + or -. Use **a+** to get automatic ASCII translations for non-commodore printers.
l followed by + or -. Use **l+** if your printer expects line feeds in addition to a carriage return.
t followed by a + or -. **t+** uses IEEE time out conventions used by COMAL.
s followed by secondary address. **s7** puts a printer in lower case mode.
d- the printer is not a disk drive.

Note that all the letters starting with a must be preceded by /. However, some letters may be omitted:

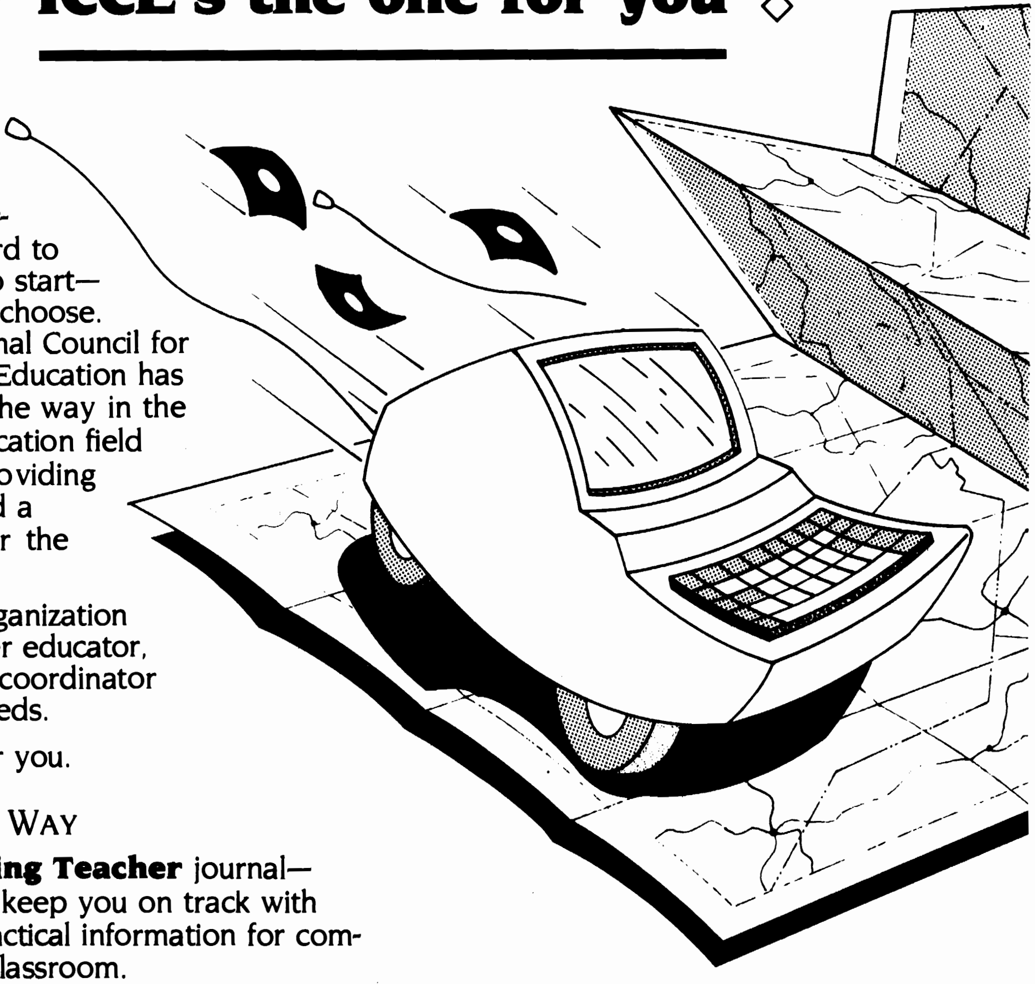
```
setprinter("u4:/a+/l+")
```

Further Printer and Plotter References:

COMAL Today #14, page 51
COMAL Today #13, page 34
COMAL Today #12, pages 10, 12, 20, 74
COMAL Today #11, page 56
COMAL Today #10, pages 7, 15, 29, 66, 72
COMAL Today #9, pages 5, 6, 18, 66, 67, 78
COMAL Today #7, pages 63, 66
COMAL Today #6, pages 11, 49, 65
COMAL Today #5, pages 18, 36, 51, 64
COMAL Today #4, pages 30, 32, 33
COMAL Today #3, pages 31, 35, 40
COMAL Today #2, pages 28, 29
COMAL Today #1, page 24 ■

Take Off With Us

◇ ICCE's the one for you ◇



With today's profusion of computer information, it's hard to know where to start—which road to choose. The International Council for Computers in Education has been guiding the way in the computer education field since 1979, providing leadership and a ground plan for the future.

It's the one organization every computer educator, administrator, coordinator or librarian needs.

It's the one for you.

Guiding the Way

The Computing Teacher journal—guaranteed to keep you on track with up-to-date, practical information for computers in the classroom.

Special Interest Groups—share information to help your special interest area grow. SIGs include computer coordinators, teacher educators, administrators and special educators, and are planned for advanced placement in computer science, community colleges and videodisc users. The quarterly **SIG Bulletin** serves as a forum for SIG information.

Booklets and Monographs point to additional information on specific topics. **ICCE Packets** provide you with teacher training materials. Members receive a 10% discount on all three.

ICCE Committees address a variety of ethical and practical issues important to you as a computer-using educator.

ICCE participates in computer education conferences throughout the world, supporting our state- and region-wide member organizations.

International Council for Computers in Education ICCE

Join the One for You.

Questions & Answers

SEVERAL QUESTIONS

Gentlemen: I have been using my COMAL 2.0 cartridge for a week now and it's great! So far beyond BASIC that there's absolutely no comparison. However, nothing's perfect, and that leads me to the following questions. - L M LaBar, Bethlehem, PA

SAVE WITH REPLACE - @

1) Commodore BASIC 2.0 is notorious for the "Save with replace" bug. COMAL's method of saving files with the "@" prefix looks like the same procedure. True?

Answer: Yes. They both use the same ROM routines. It is safest to not use the "@" save with replace at all. First initialize the disk, delete your file, then save it.

SYS 50000 BACK TO COMAL

2) Going from COMAL 2.0 by typing in the command BASIC never fails, but when I try to go back to COMAL 2.0 by typing SYS 50000, sometimes it works and sometimes not. It seems that the problem occurs most often when I have a ML routine (like *turbodisk*) in place. Disabling that ML routine doesn't help.

Answer: Some ML routines use the 50000 memory areas and thus overwrite needed code to return to COMAL. In those cases you must use a reset button or turn the computer off/on.

ABREVIATIONS

3) I have discovered the ; abbreviation for PRINT but so far have found no others, either by chance or by reading. Are there any?

Answer: Yes. Here are some shortcuts:

Short	Expanded to	
;	PRINT	
*	OF	
!	//	
NEXT	ENDFOR	
&	+	*<--- COMAL 0.14 only*
=	:=	
CHR	CHR$	
STATUS	PRINT STATUS$	

The following words are often optional:

LET, EXEC, FILE, THEN, DO, OUTPUT

Plus, you do not need to type the final quote mark in a PRINT or DATA statement if it would be the last character. COMAL will add it for you. In a similar manner, COMAL will include the name after ENDPROC, ENDFUNC, and ENDFOR, even if you don't type it (after a RUN or SCAN).

SMALL TYPE

4) My eyes are 67 years old and not as sharp as they were a half century ago. It is difficult for me to read the lowercase text portions of the *Tutorial Binder*. Could it be set in a larger type? Meanwhile, I'll use a magnifier rather than give up using COMAL.

Answer: Sorry, but Commodore produces the binder and we have no say on it's typestyle or size.

GRAPHICS SCREEN

Question: Can the graphics and turtle commands be used on the multi-color graphics screen? The commands seem to be designed with the hi-res screen in mind. - John Eldredge, Nashville, TN

More ►

Answer: Yes. The turtle and graphics commands work on both screens. Pixels are twice as wide on the multi-color screen.

C128 FEATURES FROM COMAL

Question: Is there a way to access some of the added features of the C128? As it stands, I have to choose between COMAL and C128 BASIC depending on the application. - Reed Brown, Collinsville, CT

Answer: It is now possible to access most of the added C128 features from the COMAL 2.0 cartridge with Super Chip installed. You can use Fast mode, the numeric keypad, the extra cursor keys, the other extra keys (except Caps Lock and 40/80 Display), 80 column screen, and double hi-res graphics. However, the extra memory is not accessible.

IBM PC COMAL

Question: Now that COMAL is available for the IBM PC, it would be very interesting for someone to verify just how well it works on the various "*IBM Compatible*" computers. I will probably end up with some sort of "*compatible*" in my office. It would be nice to be sure COMAL will work on it. - Bob McCauley, APO, NY

Answer: IBM PC COMAL from UniComal seems to work on most compatibles. We use it on our Zenith 151 here. We have reports that it runs fine on: IBM PC, IBM PC XT, IBM PC AT, Compaq Portable, Compaq Deskpro, Panasonic Senior Partner, Tandy 2000, and Amiga with the Transformer.

FREE MEMORY - SIZE & FREE

Question: Is there any way to check remaining memory (like BASIC **FRE(0)**)? - Reed Brown, Collinsville, CT

Answer: To check the on the amount of free memory issue this command from direct mode:

```
SIZE
```

COMAL 2.0 also has a **FREE***function included in the SYSTEM package that works from a running program. For example:*

```
USE system
leftover:=free
```

Finally, this **FREE** *function can be added to a COMAL 0.14 program using the following function:*

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

DUAL DRIVE VS TWO DRIVES

Question: The Unit to Unit File copier on page 71 of *COMAL Today #10* works fine. But I have been unable to get the COMAL 2.0 Dual Drive Copy program to work with my two 1541 drives (units 8 and 9). I get *"drive not ready"* when it asks for a disk in drive 1. - Lewis Brown, Rowayton, CT

Answer: A dual drive has two drives, numbered 0 and 1. A single drive is always drive 0. Thus your second drive is still drive 0, but unit 9. COMAL 2.0 refers to it as drive 2. There are some DOS functions (including those in the dual drive copy program) that will work fine with a dual drive, but not with two single drives. However, the COPY program on page 70 of COMAL Today #10 will work with your system. Just precede the first file name with "0:" and the second with "2:".

More ►

UN-ROM META PACKAGE

Question: The META package on *Today Disk #10* is a rommed package, so it can't be saved along with a program. Is there a way to UN-rom it so it can be saved with the program that uses it? Is there an emulation of META for COMAL 0.14? - Lewis Brown, Rowayton, CT

Answer: There is a way to "hook" the META package onto a program so it is saved along with the program. Use the link'meta procedure also on Today Disk #10. META has no equivalent in COMAL 0.14.

EMPTY SOCKET

Question: Reference to an empty socket on the COMAL 2.0 cartridge into which EPROM's may be plugged are made in the literature yet no such socket exists on my cartridge. Is my cartridge an older version or what? - William Staneski, Suffolk, VA

Answer: The first 2,000 COMAL 2.0 cartridges here in the USA and Canada used four EPROM's to hold the 64K of code for the COMAL 2.0 system. After that, the cartridges use two ROM's to hold the same 64K of code. A note was included with the first cartridges advising owners that an empty socket was not included due to lack of space on the cartridge board. If your cartridge is beige, it won't ahve an empty socket. If your cartridge case is black (with a screw in the center of its back), you have an empty socket inside the case. To access it you must open the cartridge (voiding the waranty). See page 63 of COMAL Today #13 for instructions on opening the cartridge and using the empty socket. A properly prepared EPROM (such as Super Chip) can be installed in this socket.

NESTED PROCEDURES

Question: I tried to store more than one procedure in an external procedure file, but it seems that COMAL 2.0 could only find the first one. I want to group certain external procedures into one file. - Arni Geirsson, Stanford University

Answer: You are allowed to nest procedures. This will allow you to group them into one external procedure. For example:

```
proc star'line closed
  for x=1 to 3 do
    star(x)
    line(x)
  endfor x
  //
  proc star(num)
    for num'star=1 to num do print "*",
    print
  endproc star
  //
  proc line(num)
    print "line number";num
  endproc line
  //
endproc star'line
```

If you save **star'line** *as an external procedure, both* **star** *and* **line** *procedures will be stored in that external file.*

ENHANCER 2000 PROBLEMS

Question: I can't get COMAL 0.14 to boot up. I have an **Enhancer 2000** disk drive.

Answer: the Enhancer 2000 is incompatible our fastloader. Change the boot program to skip initializing it. To do this, load the COMAL boot program. Then add the following line and save the updated boot program:

29 goto 100 ▣

COMAL Clinic

QUOTES IN FILENAME

Fred Staudaher writes: I inadvertently placed quotes around a filename I entered in response to an INPUT statement. I found that by placing 3 quotes ahead and two quotes after the filename I could DELETE the file. This saved using the disk editor to correct the directory. I have not found a reference to using double quotes within a string to obtain a quote therein.

Fred is right. You can put a quote mark into a string by typing two of them in a row. The COMAL Handbook, page 62, puts it this way: A quote mark (") can be made part of a string constant by using two consecutive quote marks (i.e., **"abc""def"** *will be read as* **abc"def***).*

2.0 INPUT DEFAULT ANSWER

It is easy to provide a default answer to INPUT questions in COMAL 2.0 programs. This means that not only will the question be printed, but a probable answer as well, with the cursor blinking on the first character of the suggested answer. To get the default reply, the user merely hits the *<return>* key. Or, they may type over the default - or use CONTROL K or SHIFT/CLR to clear out the suggested answer. This is how it's done:

```
PRINT "Enter age: 30", // comma required
INPUT AT 0,12,2: "": age //0 required
```

Remember, specifying 0 as a row or column with INPUT AT or PRINT AT means to stay in the same row or column. The PRINT statement printed both the prompt ("Enter age: ") and the default reply ("30"). The prompt is 11 characters long. The default answer starts in column 12. The 2 means to allow only 2 characters for an answer (try replying 159 and see what you get).

INTERRUPT INFORMATION

An INTERRUPT procedure is called after the current line being executed is completed. Thus, one line control structures can cause long delays in invoking high level interrupt procedures.

When a program line is done executing, COMAL checks bit 5 of EXINF ($4d). If it is set, COMAL calls the subroutine that USRQVC ($c7e2) points to. Note that the INTERRUPT procedure is not called from the IRQ interrupt. Bit 4 of EXINF flags the existence of an interrupt procedure. It is cleared when such a procedure is called and set when it is exited. This is to prevent an interrupt procedure from calling itself. However, declaring an interrupt procedure inside an interrupt procedure resets the flag. Thus, the first interrupt procedure can call another (or itself) which can call another, etc., for a kind of multi-programming. - David Martin, Ames, IA

XETEC SUPER GRAPHIX

You may be interested to know that the Xetec Super Graphix parallel printer interface has a Commodore 1525 emulation feature. Screen dumps from the Compacted Pix program work beautifully with it and my MX80. If a COMAL user needed to purchase an interface, it would certainly be one to consider. - Ed Matthews, Springfield, MO

FAST BOOT NOTE

The program *fast'boot.bas* on *Today Disk #9* is for any of our regular COMAL disks. You put this program and *ml.sizzle* on the disk with the COMAL system files, and the COMAL system is fastloaded. Better yet, see *Startup Disk* in this issue. ■

Letters

PROTECT OR NOT

Gentlemen- I had to interrupt my reading of *COMAL Today 13* this morning to give a standing ovation to Mr. Harald Nendza. Thank you for printing his letter. Other magazines may have found quieter ways to handle a letter of that type. My hat is off to you for your high standards. - John Kleczewksi, Mt Prospect, IL

Yes, Mr. Nendza's letter provoked some results. We are now happy to provide an improved and unprotected version of the Data Base program from COMAL Today #6 and #8. It seems that I am to blame for that protected program, since I missed the chance to get an unprotected copy while in California. The protection article also prompted Mike Erskine to release his Expert System on our ShareWare disk #1 in an unprotected form. I hope you will support him in this regard (it originally was to be released protected). My apologies to Robert Shingledecker. His letter follows:

Dear Len: Last issue the question of why one would protect, yet give away, a COMAL program was raised. My orginal intent was to have my name and address published within the orginal article. That way bug reports would be sure to come back to me and not be fixed and left unreported. I was unaware that *COMAL Today* usually doesn't publish an author's address.

I have no problem sharing the source code to my data base programs with all. - Robert Shingledecker, 12555-27 Euclid St, Garden Grove, CA 92640.

See the Data Base article in this issue. The full, improved and unprotected Data Base system, complete with HELP screens, is on Today Disk #14.

SUGGESTIONS

I liked Jim Ventola's idea (in *COMAL Today #11*) of sending in suggestions for packages. Here are some of my ideas:

- A printer buffer. I admit I don't know much about the way the computer communicates with the printer, but is it possible for a package to be written that would use some of the space under the cartridge as a printer buffer? Perhaps it could be done using interrupts.

- An editor/ assembler that loads from COMAL. Using other editors can be a problem when you have to call a COMAL routine (such as fndpar) that doesn't exist in BASIC, or has a different address if it does. It is no fun writing the source code in the BASIC environment, going to COMAL to test it, then going back to BASIC and re-loading the editor to fix any errors. Then you must keep repeating that sequence until you get a final working version.

Regarding your *COMAL Today #12* editorial, I agree that you should not print the longer programs in the newsletter; the disk price is so low compared to other magazines' disks that anyone who can't get the disks in their users group library can easily afford to subscribe.

I would like to see more pages in each issue of *COMAL Today*, even if it means a price increase. I read through the entire issue the day I get it and then have to wait two months for the next issue.
By the way, I sent a letter to Ahoy. I'm glad to hear that they are putting COMAL programs on their disks, and I hope they will start publishing COMAL articles in the magazine. Maybe other magazines will follow suit and we can forget about BASIC.
- David Warman, Centerville, OH

More ►

BASIC TO COMAL NOTES

Dear Sir- Sol Katz's *BASIC to COMAL* conversion program in *COMAL Today 13* is a first class demonstation of programming skills, but the first paragraph sums up its shortcomings: *"...none of the conversion necessary to get rid of the tortured logic inherent in BASIC 'spaghetti' code"*. My programming skills are strictly on the duffer level, but I have done a few conversions of simple BASIC programs and it seems to me that all-in-all it's simpler just to look at what the BASIC program does and then simply write a program in COMAL to do the same thing. No worry about *"where does the GOTO go to"* or what subterranean course the GOSUBs take. Do it in COMAL and you know where you are going and how to get there. - L M LaBar, Bethlehem, PA

Good point. I usually "convert" BASIC programs the same way - by rewriting them. The result is a clean COMAL program. However, many people want an automatic "converter" and we are happy that Sol Katz provided a good start for it. See the next letter for a similar view!

Dear Sirs- I tried to run the BASIC to COMAL program in *COMAL Today 13* but my system would not open 3 files at once. I could not figure out what was wrong, so I moved the programs to COMAL 2.0 where they worked fine.

Using the BASIC to COMAL program reminded me of the mess that results from BASIC. One program I tried to convert had a subroutine with 4 different entry points and enough GOTOs for a dozen programs. It would have been better to write a new version in COMAL but I couldn't trace the program flow. - Grant McConnell, Orangeville, Ontario

We tested the BASIC to COMAL system with our MSD drive and it worked fine in COMAL 0.14. Puzzled, we managed to find a 1541 drive to try. Then it failed as you said it would. Hmmm. We finally found the reason: the 1541 will accept 3 open files <u>only if each one has specified a drive number</u>. Now that is a very peculiar restriction for a one drive unit! Never the less, if you add a "0:" in front of each file name, it will work with COMAL 0.14. The reason it works with COMAL 2.0 is that COMAL 2.0 tacks on the drive number for you if you omit it. Thanks for spotting this for us.

ADAPT GAMES TO COMAL

Dear COMAL- Another question about programming. I'm not very oriented towards graphics, but there are some games that I would like to adapt to COMAL that could really use graphics, graphics that I don't really want to do. Are there skilled COMAL programmers who would be willing to work on programs with others, and, if so, how does one get in touch with them? Ted Mayes, 931 South Main, Maryville, MO 64468

[We've listed your address for any COMALites interested.]

APPLE COMAL NOTE

Dear Editor: With all this talk about writing Apple COMAL, why not get in touch with the Irish. The note on page 17 of *COMAL Today #2* states that they are using COMAL on Apple computers. - R Clark, North Plains, OR

The Apple COMAL referred to is a modified version of CP/M COMAL. It is a very old version of COMAL, before the COMAL Kernal standardized COMAL. Thus it does not have FUNC and ENDFUNC, and a few other

More ►

inconsistencies. It also requires a CP/M card be installed in the Apple. Plus, it costs about $230, cannot be copied, and is only available directly from its vendor, Metanic Aps, in Denmark. We hope to get an Apple COMAL that meets the standard and does not require the CP/M card.

MINI-ADA

Dear Editor: I bought *Ada Training Course* from Abacus Software. It looks like modified COMAL. Ada is supposed to be the programming language used by the Department of Defense in the future. Do you know whether or not Ada resembles COMAL or if Abacus is trying to pull something. - Bernhardt Sandler, Venice, CA

COMAL does resemble Ada. Borge Christensen, founder of COMAL, has even referred to COMAL as a "mini-Ada". I have not see the Abacus package, and thus cannot offer any other reply.

TO LIST OR NOT TO LIST

Dear Editor: I, for one, am not in favor of listing any more programs. I do take the disk subscription and it is very reasonable. I feel this is the way computer publications will have to go to give their readers what they really want and need. It really is not practical to try to type in a bunch of programs anyway. It sure is a bore to de-bug them when I type them in. - C A Barringer, Crescent City, FL

[The response from the note in the Editors Disk column in COMAL Today #12 shows that slightly more than half of our readers want fewer programs listed. To try to please everyone we will continue listing short programs, as well some long ones if we think the listing will help you.]

REQUESTS

Len - I enjoyed Jim Ventola's article in *COMAL Today #11*, page 69. His list of "wished for" programs is a good idea. Here are a few I would add to the list:

A) A sort/search package including routines for quicksort and insert sort on real, integer, and string arrays, a swap routine, and linear and binary searches.

B) A run-time program [or package] PROFILER which would produce a listing, after a program run, of the number of times each variable, procedure, or function was called. This is an extremely useful tool for program debugging and optimization.

C) A matrix package with procedures or functions for matrix multiplication, division, addition, subtraction, averaging, etc. - Kevin Quiggle, Detroit

CAD/CAM

Sirs: To turn COMAL into a CAD/CAM system, it needs a program that could generate the different "drivers" for the peripheral equipment that would be large enough to make these systems work. Then you could use different manufacturers' digitizer tablets, plotters, etc. - C M Peabody, 12226 S E 64 Place, Bellevue, WA 98006

XACTCOPY NOTE

My *XACTCOPY* procedure in *COMAL Today 13*, page 34, is designed for use with any EPSON or compatible printer, such as my Star SG10, which is about 95% Epson compatible (switch interface to linear/ non-conversion mode). - Patrick Roye, APO New York

More ►

A SMALL CHUCKLE

You have forced me to respond as if I were facing a US civil or criminal court. My number is 2223 for which no defenses are possible in your charge that my subscription terminated with No.#10. Based upon my limited integrity and (possibly yours) I have received copies five through eight plus copy ten a few days ago. The label indicates that my last issue is #10.

I have always felt that the Europeans possessed the greatest integrity second only to northern neighbors of mine.

The integrity of your southern neighbor is comparable to South Korea. Nevertheless, people tend to be real regardless of their origin.

Now that I'm through intimidiating those things which are of value, (namely this computer and your service) I would like to finish with not satire but sincerity.

My appeal is forthwright...

I wish to denounce my US citizenship for otherwise my integrity will be lost. If in any way you perceive some logic to this, please forward this to what ever diplomatic offices available within your area.

PS: My appologies for confusing you with Canadians. Galen Foreman, Hutchinson, KS

[Hmmm... not really knowing what else to do, we sent another copy of COMAL Today #9 to Mr. Foreman. Hope he renews! Any comments? I told you we get interesting mail.] ■

Bug Fixes

SPECIFY DRIVE NUMBER

The 1541 disk drive will allow three files to be open at once only if a drive number is specified for each of them. This restriction does not apply to MSD disk drives. COMAL 2.0 always tacks on the current drive number for you if you do not include it, so the problem is bypassed. Grant McConnell pointed out that our *BASIC TO COMAL* programs on page 42 of *COMAL Today #13* do not include the "0:" drive specification. To use the programs in COMAL 0.14 with a 1541 drive, you must add "0:" at the start of each filename.

SIGDIG UPDATE

The *sigdig* program on page 59 of *COMAL Today #13* can have problems with certain numbers. This problem was corrected in time for *Today Disk #13*. The new version below is shown below:

```
func sigdig(n$) closed
  while n$(1)="0" do n$:=n$(2:len(n$))
  dec'pt:="." in n$
  if dec'pt=len(n$) then dec'pt:=0
  if dec'pt then
    n$:=n$(1:dec'pt-1)+n$(dec'pt+1:len(n$))
    while n$(1)="0" do n$:=n$(2:len(n$))
  else
    while n$(len(n$))="0" do n$:=n$(1:len(n$)-1)
  endif
  if n$(len(n$))="." then return len(n$)-1
  return len(n$)
endfunc sigdig
```

SINGLE FILE COPY NOTE

Bob Hoerter reports that the *single'file'copy* program on the 2.0 side of *Today Disk #9* doesn't work. Use the *copy* program on *Today Disk #10* instead, or for copying large files, see the *Single File Copier* article in this issue. ■

COMAL Structures Case Statements

by Richard Bain

The **CASE** statement is like the **IF** statement except that **CASE** is more flexible, similar to a multiple choice question. Here is a simple example to set the cursor color within a program by asking the user to enter the color by name (note that COMAL 2.0 has different names for the color commands).

```
dim color$ of 16
// use graphics // COMAL 2.0 only
input "what's your favorite color?":color$
case color$ of
when "black"
  pencolor 0 // textcolor(0) in 2.0
  border 0 // textborder(0) in 2.0
when "white"
  pencolor 1 // textcolor(1) in 2.0
  border 1 // textborder(1) in 2.0
// put check for other colors here
when "gray","grey"
  pencolor 15 // textcolor(15) in 2.0
  border 15 // textborder(15) in 2.0
otherwise
  print "I don't know that color"
endcase
```

This **CASE** statement may be long (if all 16 colors are checked), but it will help make your program more *user friendly*. It will also make your program readable.

The first line in the above example is important; all strings must be dimensioned before they are used. On the fifth line, **color$** is compared to "**black**". This is the first "case". If there is a match, COMAL executes the lines below this **WHEN** statement. If the first WHEN cases do not include a match, COMAL continues to check for a match in the next WHEN statement. If no match is found in any of the WHEN cases, COMAL then executes the statements following the OTHERWISE. Once a block of statements are executed from within a WHEN or OTHERWISE section, COMAL skips down to the line after the ENDCASE. Only one block of statements in a CASE structure is executed. Note that you can compare **color$** to several items in a **WHEN** statement. You can allow for multiple spellings.

In the third to last line, **OTHERWISE** is important. If the user doesn't want the color changed, a simple *<return>* will allow the program to continue without any unwanted changes. If the **OTHERWISE** section were missing, hitting *<return>* alone, or entering an unknown color, would cause an error (no match found).

There are a few alternatives to using a case statement. You could use 16 one line **IF** statements to simulate the above example (one for each color).

```
if color$ = "light grey" or color$ = "grey" then pencolor 15
```

You could also make the user enter the color by number. This would make the program shorter, but not as *user friendly*. Asking for **red** is generally more natural than asking for **2**.

Another common use for the **CASE** statement is in choosing an item from a menu. The **CASE** statement may be placed within a **REPEAT** loop to allow for several consecutive choices.

A **REPEAT** loop is used in the next example instead of a **WHILE** loop because you always want to see the menu at least once. Typing a **4** will let you exit the loop. **Game**, **multiply**, and **help'checking** are procedure calls. The procedures must appear somewhere else in the program or be supplied as **EXTERNAL** procedures. Here, the case conditional variable (**choice**) is a number, while in the previous example it

More ►

was a string. This shows even more of the flexibility of the **CASE** statement.

```
repeat
  print chr$(147) // clearscreen
  print "1 - play a game"
  print "2 - practice multiplication"
  print "3 - balance check book"
  print "4 - quit"
  input choice
  case choice of
  when 1
    game
  when 2
    multiply
  when 3
    help'checking
  when 4
    null // quit option - do nothing
  otherwise
    print "type 1, 2, 3, or 4 only"
  endcase
until choice = 4
```

COMAL has a wide variety of structures. The **CASE** statement is only one of them. The beginning programmer should write short programs trying out the different features. Editing other people's programs (such as the ones found in this magazine and on the matching disk) can help even more. After gaining programming experience and becoming familiar with structures and structured programming, you can wisely choose the best structures for your programming needs. Even if you don't want to try every feature now, COMAL has the flexibility to let you accomplish your goals in a satisfying and creative way.

Further References:

COMAL Structures: While Loops, COMAL Today #13, page 16
Making Decisions with IF-THEN-ELSE, COMAL Today #10, page 58 ■

Best Sellers

We are happy to see one of the original COMAL text books, *Foundations in Computer Studies*, back in the charts - #3 for August. *COMAL Today - The INDEX* is already a best seller, and it isn't even released yet. With over 800 pages in the first 12 *COMAL Todays*, you really do need an index.

June 1986 (final)

#1 - **Introduction to Computer Programming**
by J William Leary
#2 - **Introduction's Answer Book**
by J William Leary
#3 - **Cartridge Tutorial Binder**
by Frank Bason & Leo Hojsholt
#4 - **COMAL Handbook**
by Len Lindsay
#5 - **COMAL Workbook**
by Gordon Shigley

July 1986

#1 - **Introduction to Computer Programming**
by J William Leary
#2 - **COMAL From A to Z**
by Borge Christianson
#3 - **Cartridge Tutorial Binder**
by Frank Bason & Leo Hojsholt
- **Cartridge Graphics & Sound**
by Captain COMAL's Friends
#4 - **Introduction's Answer Book**
by J William Leary
#5 - **COMAL Handbook**
by Len Lindsay

August 1986

#1 - **COMAL From A to Z**
by Borge Christianson
#2 - **COMAL Workbook**
by Gordon Shigley
#3 - **Foundations in Computer Studies**
by John Kelly
#4 - **COMAL Today - The INDEX**
by Kevin Quiggle
#5 - **COMAL Handbook**
by Len Lindsay ■

Petals Around The Rose

By Bill Inhelder

About seven years ago when microcomputers were in their infancy, a unique computer game called *Petals Around The Rose* was circulating among programmers. Because the game is unfamiliar to many newcomers to the ranks of personal computer users and for purely nostalgic reasons, I thought it might be appropriate to present a reprise of the game.

The object of the game is to determine how the game is played! Five dice are tossed and the results are displayed graphically on the screen. The player must guess a number which is in some way related to the toss of the dice. The computer will then give the correct response. The object of the game is to discover that relationship so that the player's response will match the computer's response. Only three hints are given the player:

1. The name of the game is **Petals Around The Rose**.
2. The solution is always an even number.
3. The computer always gives the correct result.

Six correct responses in a row will demonstrate that the player knows how the game is played. The player is then pledged to secrecy and is awarded a personalized certificate of membership in The Society of Petals Around the Rose. The certificate may optionally be copied to your printer.

It may be helpful to play the game with pencil and paper handy, but avoid complex mathematical formulas. The solution is truly simple! Keep the hints in mind, especially the first one. Try to make sense out of it. Be warned: many brilliant and learned persons have failed, while others pass off the solution as trivial.

Once you have solved the problem try it on your friends and you can observe sympathetically or smugly, depending on your nature, your friends' level of frustration.

Listing of the lines of the program is inhibited. This was done so that a frustrated player would not be tempted to peek at the solution to the game by reading the program listing. This does not prevent the program from being copied to other disks.

My thanks to Borge Christensen for the technique of inhibiting and permitting the listing of a program in *Rod the Roadman* in *COMAL Today #9.*

[Editors note: this program originally included a routine to print the certifacate in an alternate font. This had to be eliminated to allow space on Today Disk #14 for additional programs.]

```
**********************************
*                                *
*   NATIONAL SOCIETAL ORDER OF   *
*                                *
*     PETALS AROUND THE ROSE     *
*                                *
*          Confers Upon          *
*                                *
*                                *
*     Herbert J Greensmith       *
*                                *
*                                *
* This Certificate of Membership *
*                                *
* With All Rights and Privileges *
*                                *
**********************************
```

Program Outliner

by Len Lindsay

"Aldebaran's $97 utility, Source Print, offers a structured outline format feature that automatically draws connecting lines between program block elements."
-- PC Magazine, Sept 16, 1986, page 63

While trying to keep up to date, I manage to read literally dozens of computer magazines and newsletters each week. The notice reprinted above caught my eye. Program outlining is a very nice idea. Why not give *COMAL Today* readers a program outliner?

I sat down, and in 1 hour had a program that would outline both COMAL 0.14 and COMAL 2.0 programs that were LISTed to disk. The COMAL 0.14 programs must be listed with indentations (see page 3 for the modification to COMAL). The program is listed below - output from itself! It doesn't have the fancy features of the IBM PC program, but it is short and readable.

```
    // delete "program'outliner"
    // save   "program'outliner"
    //  by Len Lindsay
    DIM space$ OF 3, line$ OF 3
    DIM top$ OF 3, bottom$ OF 3
    DIM middle$ OF 3, filename$ OF 20
    DIM out$ OF 20, this'line$ OF 120
    DIM next'line$ OF 120
    init
    printout
    //
==> PROC init
!       disk'indent:=0
!       // set at first structure top
!       space$:="   " // 3 spaces
!       line$:="!  " // ! and 2 spaces
!       top$:="==>"
!       middle$:="+->"
!       bottom$:="-->"
!       PRINT CHR$(147),CHR$(14)
!       PRINT TAB(8),"Program OUTLINE Processor"
!       PRINT
!       PRINT "This program takes a listed program on"
!       PRINT "disk and prints it to the screen (ds:),"
!       PRINT "printer (lp:), or disk file (0:filename)",
!       PRINT "with the different structures outlined."
!       PRINT
!       INPUT "listed program filename? ": filename$
!       INPUT "output location? ": out$
--> ENDPROC init
==> PROC printout
!       OPEN FILE 2,filename$,READ
!       SELECT OUTPUT out$
!       INPUT FILE 2: this'line$
!       fix'line(this'line$)
!       this'indent:=indent(this'line$)
!       last'indent:=this'indent
!   ==> WHILE NOT EOF(2) DO
!   !       INPUT FILE 2: next'line$
!   !       fix'line(next'line$)
!   !       next'indent:=indent(next'line$)
!   !       preline
!   !       PRINT this'line$
!   !       this'line$:=next'line$
!   !       last'indent:=this'indent
!   !       this'indent:=next'indent
!   --> ENDWHILE
!       next'indent:=indent(this'line$)
!       preline
!       PRINT this'line$ // last line in file
!       CLOSE FILE 2
!       SELECT OUTPUT "ds:"
--> ENDPROC printout
    //
==> FUNC indent(REF text$) // first space is not indent
!       level:=0
!   ==> WHILE LEN(text$)>1 DO
!   !   ==> IF text$(2:2)=" " THEN
!   !   !       level:+1
!   !   !       text$:=text$(2:LEN(text$))
!   !   +-> ELSE
!   !   !       RETURN level
!   !   --> ENDIF
!   --> ENDWHILE
!       // text$:=text$+"//" // optional tack on blank
!       RETURN this'indent // blank line at same level
--> ENDFUNC indent
    //
==> PROC fix'line(REF text$) // remove line number
!       text$:=text$(5:LEN(text$))
--> ENDPROC fix'line
    //
==> PROC preline
!       prefix
!   ==> IF this'indent<last'indent AND this'indent<
        next'indent THEN // wrap line
!   !       PRINT middle$,
!   +-> ELIF this'indent<next'indent THEN
!   !       PRINT top$,
!   +-> ELIF this'indent<last'indent THEN
!   !       PRINT bottom$,
!   +-> ELSE
!   !       PRINT space$,
!   --> ENDIF
--> ENDPROC preline
    //
==> PROC prefix
!   ==> IF disk'indent THEN
!   !   ==> FOR temp=1 TO this'indent STEP disk'indent DO
!   !   !       PRINT line$;
!   !   --> ENDFOR temp
!   +-> ELSE
!   !       disk'indent:=next'indent-this'indent
!   --> ENDIF
--> ENDPROC prefix ■
```

Right Turn Only

by Phil and Phyrne Bacon

Right'turn'only is a program which draws a boxtree by drawing a continuous line with many right turns.

At level 0, there are 4 right turns, at level 1 there are 12 right turns, and so on, until at level six, 2916 right turns.

Right'turn'only is on *Today Disk #14.*

COMAL 2.0 version:

```
// delete "right'turn"
// save  "right'turn"
// by Phil and Phyrne Bacon
//   (904) 376-9539
//
// This program is in the
//    public domain
init
get'level
main'program
pause
textscreen
END "Done."
//
PROC get'level
  LOOP
    TRAP
      PAGE
      PRINT "Draw to which level";
      PRINT "(0,1,2,3,4,5,6)? 5"157"",
      INPUT AT 0,0,1: "": endlevel
      EXIT WHEN endlevel<7
    HANDLER
      bell(1)
    ENDTRAP
  ENDLOOP
ENDPROC get'level
//
PROC init
  USE system
  USE graphics
  graphicscreen(0)
  textscreen
  background(1)
  border(1)
  pencolor(13)
ENDPROC init
//
PROC main'program
  fullscreen
  IF endlevel MOD 2 THEN
    moveto(184,4) // odd number
  ELSE
    moveto(132,4) // even number
  ENDIF
  length:=96; level:=0
  box(level,length)
  forward(length)
  IF endlevel<>0 THEN // 4th corner
    box(level+1,length/2)
  ENDIF
ENDPROC main'program
//
PROC box(level,length)
  IF level<>endlevel THEN
    forward(length)
    box(level+1,length/2)
    forward(length)
    box(level+1,length/2)
    forward(length)
    box(level+1,length/2)
  ELSE
    IF level=6 THEN length:=length/2
    forward(length)
    right(90)
    forward(length)
    right(90)
    forward(length)
    right(90)
  ENDIF
ENDPROC box
//
PROC pause
  WHILE KEY$<>CHR$(0) DO NULL
  WHILE KEY$=CHR$(0) DO NULL
ENDPROC pause
```

COMAL 0.14 version:

```
// delete "0:right'turn"
// by Phil and Phyrne Bacon
// save  "0:right'turn"
//   (904) 376-9539
// this program is in the
//    public domain
//
get'level
init
main'program
pause
settext
print chr$(14),"Done."
end
//
proc get'level
 repeat
  print chr$(147),chr$(14)
  print "Draw to which level";
  print "(0,1,2,3,4,5,6)?";
  input "5"+chr$(157): endlevel
 until endlevel>=0 and endlevel<7
endproc get'level
//
proc init
 background (1)
 border (1)
 pencolor (13)
 setgraphic (0)
 hideturtle
endproc init
//
proc main'program
 if endlevel/2<>int(endlevel/2) then
  moveto 184,4
 else
  moveto 132,4
 endif
 length:=96; level:=0
 box(level,length)
 forward (length)
 if endlevel<>0 then // 4th corner
  box(level+1,length/2)
 endif
endproc main'program
//
proc box(level,length)
 if level<>endlevel then
  forward (length)
  box(level+1,length/2)
  forward (length)
  box(level+1,length/2)
  forward (length)
  box(level+1,length/2)
 else
  if level=6 then length:=length/2
  forward (length)
  right (90)
  forward (length)
  right (90)
  forward (length)
  right (90)
 endif
endproc box
//
proc pause
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
endproc pause
```

Single File Copying

by David Stidolph

There are many times when you may wish to copy just a couple files from one disk onto another. To help you with this task, we provided a program, *single file copy*, on *Today Disk #1 and #2.* It is a very nice program, but can't copy a large program file like *c64 comal 0.14.*

Now we are providing another single file copy program. It is not as friendly as the first one, but this one will copy files up to 136 blocks long. Thus, it will copy the *c64 comal 0.14* file (131 blocks). The new program, called *copyfile2.basic*, is on *Today Disk #14.* Maximum memory was needed for copying a large file, so the program was written in BASIC, with a machine language copy routine. Do <u>not</u> load it into COMAL. Here is how to use it:

1) Turn computer system on.

2) Insert *Today Disk #14* into drive.

3) Enter these commands:
   load "copyfile2.basic",8
   run

4) The screen clears. The program asks:
   Please enter filename with
   ,s for SEQ or ,p for PRG
   after the name: ?

5) Remove the disk from the drive.

6) Insert the disk with the file you wish to copy on it into the drive.

7) Type in the exact name of the file you wish to copy. If it is a program file (**PRG** in the disk catalog) add a **,p** after its name. For example:
   c64 comal 0.14,p

   If it is a sequential file (**SEQ** in the disk catalog) add a **,s** after its name. For example:
   comalerrors,s

8) The program initializes the disk and reads in the entire file you specified. This message appears on the screen:
   Reading file...

9) If it encounters a disk error it will stop and print:
   Disk problem!

   If the file is too big (over 136 blocks) it will stop and print:
   File too large!

   If all is OK it will prompt you to switch disks:
   Insert disk to write file to
   and press RETURN ?

10) If all was OK, take out the original disk and insert the disk you wish to write the file to. Then hit the *<return>* key.

11) While the file is copied onto the new disk this message is displayed on your screen:
   Writing file...

12) When the file is completely copied, the program ends with this message:
   All done.

13) If you wish to copy another file, enter the command: <u>**run**</u> and continue at step number 4.

14) When done copying files, turn off your computer. Turn it back on if you wish to run other programs. ■

COMAL 0.14 Startup Disk

Does this sound familiar? You have had COMAL 0.14 for a short time now and want to start writing your own programs. Of course, once you have written that first program, you will want to save it too. Maybe you got your first COMAL disk from your local users group, *Ahoy!* magazine, or COMAL Users Group, U.S.A., Limited.

However, there are two problems writing on these disks. First, you should **never** write to an original disk made on a disk drive other than your own. Back it up first and write to the copy (*COMAL Today #10*, page 60, explains why). Second, these disks tend to be rather full and don't have room for your programs.

To use COMAL 0.14, you must have it on a disk to load into your computer. COMAL has been released under a few different file names, but all versions are exactly 131 blocks long and COMAL is usually included in the file name. The program has not been changed even if its name has. It is called *c64 comal 0.14* on *Today Disk #14* (that is it's official name). This one file is the absolute minimum that must be on your COMAL 0.14 start up disk. It can be accessed from BASIC by loading and running it. However, this will not give you COMAL with expanded memory and fast error messages in RAM (that is why we include a boot program too).

People who use COMAL, quickly learn about the user friendly error messages. Did you know that these error messages are not included directly within the language? A second file called *comalerrors* contains these messages, allowing the messages to be in any language necessary (such as French). This file also belongs on your COMAL startup disk.

The above two files are enough for a working COMAL system, but three more files are needed to make COMAL as user friendly as possible. Phil Bacon wrote *ml.sizzle* to reduce the time required to load COMAL. John McCoy wrote a procedure to expand the amount of memory available to COMAL users. Robert Ross wrote a routine to let the error messages reside in unused memory under the I/O block at $d000.

[Originally, the error messages had to be loaded in from disk each time an error occured. Next the messages were stored in sprite memory reducing the number of possible sprite images. Now, this restriction no longer applies.]

The routines to expand RAM and load the error messages under the I/O are included in the program called *hi* on *Today Disk #14*. One more file is used to make everything work together. *Boot c64 comal*, should be the first file copied to the startup disk. This is the program that you LOAD and RUN from BASIC to startup the entire COMAL system.

These files are the first five files on *Today Disk #14*. They should be copied to any disk you plan to use with COMAL.

Since COMAL disks are not copy protected, you can copy the files using any standard copy program you have. Public Domain copy programs may be obtained from local user groups. Note that some copy programs cannot copy a file as large as 131 blocks. To overcome this limit, we are providing a special, no frills file copier. It is called *copyfile2.basic* and also is on *Today Disk #14*. It uses a machine language copy routine so it can copy a file as fast as your drive can read and write it. Instructions for using it are included in the article just before this one.

More ►

MAKE A STARTUP DISK

This is a quick summary on how to create your very own COMAL 0.14 start up disk.

1) Turn on system.
Turn on your computer and disk drive. Remain in BASIC. Don't load and run any other programs yet.

2) Prepare new disk.
To format a blank disk for use as your start up disk, insert the blank disk into your disk drive and type in this command (use the digit zero in n0):

```
open 1,8,15,"n0:startup,ss":close1
```

3) Label the disk.
Remove the disk from the drive. Prepare a label for the disk. Call the disk **COMAL Startup - Master Disk.** Make the label first, then stick it onto the upper left corner of your disk.

4) RUN the File Copier.
Insert *Today Disk #14* into your disk drive, label side up. Run the file copier by typing these commands:

```
load"copyfile2.basic",8
run
```

5) First copy the boot file.
Reply to the filename prompt:

```
boot c64 comal,p
```

After it is read in, you will be prompted to remove the disk and insert the disk to copy it to. Take the disk out, insert your new startup disk, and hit the *<return>* key.

6) Copy 4 other files.
After copying the boot file, continue by copying the 4 other files. The copy program will end after copying a file. To copy another file, just type: RUN. Use these replies to the filename prompt:

```
c64 comal 0.14,p
hi,p
ml.sizzle,p
comalerrors,s
```

7) Remove the disk.
Remove the disk from the drive. It is your "master" COMAL disk.

8) Backup the disk.
You should use a **disk backup** program to duplicate your start up disk before you use it. Use any disk backup program, or see *COMAL Today #10* page 60, for directions.

9) Use copies only.
Every time you need a new COMAL disk, you can just make a duplicate of your newly created COMAL start up disk. Duplicating the disk is much faster than creating a new one, file by file. You can save any COMAL programs that you write onto these disks. Each disk will then contain the complete COMAL system along with your programs.

You can **give** copies of your startup disk to friends, Users Group, or School, but you may not **sell** them. You are allowed to sell your own COMAL programs, and include the full COMAL 0.14 system on your disks, as long as you specify that you are selling your programs. You include the COMAL 0.14 system at no extra charge. ▣

Listerine

by Will Bow

This program takes a listed COMAL program from disk and creates separate files for each procedure and function. It works unchanged in either version of COMAL, and will work on any listed program. It is on *Today Disk #14*.

Listerine allows you to break any program into its component procedures and functions so they can be used by other programs. This is a simple automated way to start a procedures library from programs that you already have.

Listerine is smart and can handle the nested procedures of COMAL 2.0. It doesn't separate out the nested procedures, but keeps them nested in the outer procedure.

The program is listed here with the structures outlined (see *Program Outliner*, page 17). Originally *listerine* was written in COMAL 2.0. It was rewritten to work with COMAL 0.14 as well.

```
// delete "listerine"
// save   "listerine"
//  by Will Bow
// version 2.1  6-12-85
// coverted to work with both
// comal versions by Captain COMAL
// on 8-28-86
DIM text$ OF 160, answer$ OF 1
DIM in'file'name$ OF 18, inst$ OF 40
DIM out'file'name$ OF 18
DIM name$ OF 80, space$ OF 40
DIM p'f'type$ OF 15, type$ OF 40
space$(1:40):=""
//
title'page
get'filename
read'in'listed'file
PRINT "Have a Nice Day !!"
END
```

```
    //
==> PROC read'in'listed'file
!       OPEN FILE 10,in'file'name$,READ
!   ==> WHILE NOT EOF(10) DO
!   !       INPUT FILE 10: text$
!   !   ==> IF "//" IN text$ THEN
!   !   !       text$:=text$(1:("//" IN text$)-1)
!   !   --> ENDIF
!   !   ==> IF "proc " IN text$ OR "PROC " IN text$ THEN
!   !   !   ==> IF NOT """" IN text$ THEN
!   !   !   !       make'proc'name
!   !   !   !       list'proc'to'disk
!   !   !   --> ENDIF
!   !   +-> ELIF " func " IN text$ OR " FUNC " IN text$
!   !   !       make'func'name
!   !   !       list'func'to'disk
!   !   --> ENDIF
!   --> ENDWHILE
!       CLOSE FILE 10
--> ENDPROC read'in'listed'file
    //
==> PROC list'proc'to'disk
!       OPEN FILE 20,out'file'name$,WRITE
!       PRINT FILE 20: text$
!   ==> REPEAT
!   !       INPUT FILE 10: text$
!   !       PRINT FILE 20: text$
!   !       text$:=text$+" "
!   --> UNTIL "endproc "+name$+" " IN text$ OR
!       "ENDPROC "+name$+" " IN text$ //wrap line
!       CLOSE FILE 20
--> ENDPROC list'proc'to'disk
    //
==> PROC make'proc'name
!       IF "proc " IN text$ THEN type$:=" proc "
!       IF "PROC " IN text$ THEN type$:=" PROC "
!       p:=(type$ IN text$)+6
!       name$:=text$(p:LEN(text$))
!       par:="(" IN name$; sp:=" " IN name$
!   ==> IF par>0 AND par<sp THEN
!   !       name$:=name$(1:("(" IN name$)-1)
!   +-> ELSE
!   !       name$:=name$(1:(" " IN name$)-1)
!   --> ENDIF
!       out'file'name$:="0:proc."+name$
!       PRINT out'file'name$
--> ENDPROC make'proc'name
    //
==> PROC title'page
!       PRINT CHR$(147),CHR$(14),CHR$(18),
!       move'cursor(5,10)
!       PRINT " L I S T E R I N E "
!       PRINT
!       PRINT TAB(14),"by Will Bow"
!       PRINT CHR$(13)
!       PRINT "Do you need instructions ? <y/n> n",
!       INPUT CHR$(157): answer$
!       IF answer$="y" THEN instructions
--> ENDPROC title'page
    //
==> PROC instructions
!       PRINT CHR$(147)
!       move'cursor(5,1)
```

More ►

```
!    ==> WHILE NOT EOD DO
!    !       READ inst$
!    !       PRINT inst$
!    --> ENDWHILE
!        move'cursor(20,7)
!        PRINT CHR$(18),"PRESS ANY KEY TO CONTINUE"
!        WHILE KEY$>CHR$(0) DO NULL
!        WHILE KEY$=CHR$(0) DO NULL
!        move'cursor(20,7)
!        PRINT space$(1:30)
--> ENDPROC instructions
    //
    DATA "LISTERINE reads in a LISTED COMAL"
    DATA "program file from the disk and then"
    DATA "writes each PROC in the file back to"
    DATA "the disk. Each PROC's filename is the"
    DATA "PROC name. This will allow you to"
    DATA "break up your program and MERGE the"
    DATA "PROCs back in any order you choose."
    DATA "  WARNING:"
    DATA "This program WRITEs to the disk. Make"
    DATA "sure you have enough FREE BLOCKS on"
    DATA "the disk before you begin."
    //
==> PROC get'filename
!        move'cursor(20,3)
!        INPUT "What's the filename: ": in'file'name$
--> ENDPROC get'filename
    //
==> PROC make'func'name
!        IF " func " IN text$ THEN type$:=" func "
!        IF " FUNC " IN text$ THEN type$:=" FUNC "
!        p:=(type$ IN text$)+6
!        name$:=text$(p:LEN(text$))
!        par:="(" IN name$; sp:=" " IN name$
!    ==> IF par>0 AND par<sp THEN
!    !       name$:=name$(1:("(" IN name$)-1)
!    +-> ELSE
!    !       name$:=name$(1:(" " IN name$)-1)
!    --> ENDIF
!        out'file'name$:="0:func."+name$
!        PRINT out'file'name$
--> ENDPROC make'func'name
    //
==> PROC list'func'to'disk
!        OPEN FILE 20,out'file'name$,WRITE
!        PRINT FILE 20: text$
!    ==> REPEAT
!    !       INPUT FILE 10: text$
!    !       PRINT FILE 20: text$
!    !       text$:=text$+" "
!    --> UNTIL "endfunc "+name$ IN text$ OR
!        "ENDFUNC "+name$ IN text$ //wrap line
!        CLOSE FILE 20
--> ENDPROC list'func'to'disk
    //
==> PROC move'cursor(row,col) CLOSED
!        row:-1; col:-1
!        POKE 209,(row*40+1024) MOD 256
!        POKE 210,(row*40+1024) DIV 256
!        POKE 211,col
!        POKE 214,row
--> ENDPROC move'cursor
```

JOIN THE ON-LINE COMMODORE® USER GROUP.

Imagine being part of a nationwide on-line user group. With new QuantumLink, you can instantly exchange ideas, information and software with Commodore users everywhere, and participate in live discussions with Commodore experts.

And you can participate in conferences held by Len Lindsay, access COMAL public domain programs, and have your questions answered by other Comalites. You can even share your public domain COMAL programs with others.

These are just a few of the hundreds of features available. If you already have a modem, you can register on-line for a free software kit and trial subscription. Hook up and call **800-833-9400**. If you need a modem, call QuantumLink Customer Service at 800-392-8200.

QUANTUMLINK™ The Commodore® Connection

Subscribe Today!

To The Southeastern United States fastest growing Commodore Users Group tabloid newspaper

SPARKPLUG!

Published monthly by the Spartanburg Commodore Users Group (SPARCUG)

☐ $12.00 Per Year U.S.

☐ $1.00 For Sample Copy

☐ $20.00 Per Year U.S. SPARCUG Associate Membership

NAME: __________

ADDRESS: __________

CITY: __________ STATE: ____ ZIP: ____

Send Check or Money Order To:

SPARCUG

P. O. Box 319
Spartanburg, S.C. 29304.

Multi function Graphics

by Lowell Toms

Recently, I ran across a new hand held calculator which was able to plot functions on a slightly oversized LCD screen. Well, it was pretty neat, but the last thing I need is another calculator. While a graphing routine may be a chore to program in BASIC, COMAL is a natural for the task. Before you skip on to the next program while mumbling *who needs another graphing routine*, you should know that this one has some different capabilities. The routine can plot up to six equations **(y=f(x))** on the same screen. These equations are entered and deleted while the program is running. The range and origin of the X and Y axes are easily modified from within the program. Finally, a joystick controlled routine prints the coordinates of any interesting point to the screen. This eliminates the need for an extensive grid, and gives much more accurate results than the *eyeball* approach. Hopefully, I've piqued your curiosity enough that you're ready to load the program off *Today Disk #14*.

Graphic'solutions includes four different processes. When the initial menu screen appears you may select to enter a function, define the axes, plot a function, or find a point on your plot. Select **1** and a new menu appears which allows you to enter a function, view previously entered functions, clear all functions, or return to the main menu. Select **1** again and you are asked in which position **(1-6)** you want to place your function. If you hit *<return>*, the function is automatically placed in the first open position, (enter another number to overwrite a previous entry). Now, you are asked to enter your function:

```
Y=X^4+5*X^3-7*X^2-29*X+30
```

The computer pauses for a moment while the formula is entered. If you've made a syntax error in the formula, the program stops and an error message is displayed. Just correct the error and hit *<return>* until each displayed line is entered and **RUN** sends you back to the main menu.

Now, let's set the axes. Select **2**, and you are asked to specify the origin. For this case, hit *<return>* and the default values of **0,0** are entered. Next, you're asked to set the *x tic marks*. Hit *<return>* and the default value of **1** is entered. For the *y tic marks*, enter **10** and hit *<return>*. The next question asks if you want the origin and *tic* values displayed on the graph. Hit *<return>*, and the values will be displayed. The main menu is displayed again, with the new values shown below the <u>set axes</u> selection.

It's finally time to plot. Select **3** from the menu, and then select **1** from the plot menu. Enter **1** (or **-1** for faster, but cruder plot) of function **1**. When the function plot is complete, a **>** appears in the upper left corner. Hit any key, and you will return to the plot menu. Type **3** to return to the main menu, and select **4**.

Selection **4** is the joystick point finding routine. Plug your joystick into port 2, select **1**, and hit *<return>*. The plot reappears with the turtle present at the screen center. The turtle speed can be varied by pressing the **+** and **-** keys. Move the turtle to the point of interest and hit the fire button. The point's coordinates are displayed in the lower right corner.

You're certainly not limited to graphing polynomials. Set the axes back to their default values (type **2** from the main menu and then hit *<return>* to all the prompts).

More ►

Enter Y=TAN(X) keeping the same axes as before. Next try Y=1/X. This equation will create an error at x=0 unless you answer y to the avoid a point? query in the plot section and enter 0 as the point to avoid. Some functions such as **SQR** don't digest negative numbers very well so you have the option of specifying positive x values only. Of course you could just shift the axes, but you may wish to plot another function on the same screen without a shifted origin. The plotting option also allows automatic plotting of all the entered functions by selecting 7 (or -7). You must press a key when the > appears, but the routine starts plotting the next function instead of returning to the menu.

I was forced to make some compromises in the program due to the 12k available memory (and my inexperience with COMAL). The most serious compromise was the elimination of the bulk of the code comments. Even with minimal comments, the free memory available is only about 1/2k, so you may want to watch the memory size if you enter large formulas. *(You must have an expanded memory version of COMAL 0.14 to run this program. Today Disk #14 expands memory.)*

Another warning is also in order. Do not renumber the first section of the program (lines 1-999). This section is the area where the dynamic keyboard routine modifies program lines. There is no screen dump utility built into the program, but there are plenty of dump routines available (try the *Utilities #2*).

I hope you enjoy the program, and I would like to thank Dick Klingens for his **STR$** and VAL routine listed in *COMAL Today #12*. ■

Dual screen

by Craig Hardy

The purpose of this package is to copy the contents of the graphics screen (hi-res or multi-color) and its color to a free memory area in 1/6 of a second. It restores the screen when the appropriate command is issued. *Pkg.dualscreen* is on *Today Disk #14.*

To use the package:

```
LINK "pkg.dualscreen"
USE dualscreen
```

This package will add two commands:

```
  getscrn //copy the bitmap and colors
  putscrn //restore the original screen
```

This package could be used in a graphics editor program as an *oops* key or you can use the package when you would prefer to load two screens in at the beginning of your program rather than disrupt the program with a second **LOADSCREEN**. Incidentally, you do not have to have the graphics screen active to do this. You could be displaying a textscreen. In that case the sequence of commands should be:

```
USE dualscreen
USE graphics
loadscreen (filename$) //load 1st screen
getscrn //save first screen
loadscreen (filename2$)//load 2nd screen
fullscreen //display 2nd screen

   <<< Program continues here >>>

//Now, to display 1st screen you loaded
//you don't have to specify
//hi-res or multicolor.
border(0) //specify border color#
background(0) //specify background#
putscrn //restore screen
fullscreen //display screen
```
■

Student Programs

by Larry Winckles

The following programs are the best of the work from my 11th grade COMAL class. Most of the programs are self explanatory, but I would like to comment on a few of them.

As a culmination of our study of graphics and input devices, I assigned the students the task of writing a program which made use of both graphics and either the Koala pad, joystick, or Muppet Learning Keys. I wrote a short demonstration program for interpreting input from the Muppet Learning Keys. This program is on *Today Disk #14*.

Several students chose to try their hand at graphic drawing programs, with varying degrees of success. Bill Howard, who wrote *bill'paint*, has a Macintosh computer at home, so he created icons as a means of interacting with his program. His greatest frustration with his program was not being able to easily add the ability to load/save graphic screens.

I would like to point out that *designs* makes use of the **PLOTTEXT** command on the multicolor screen. According to *Commodore 64 Graphics with COMAL*, page 99, this should not work. Do you have any ideas as to why it does? I believe it has something to do with clearing out the underlying textscreen immediately prior to plotting the text.

[The textscreen and multi-color graphics screen share color information, so clearing the text screen should help in plotting characters. Some multi-color characters look better than others; it seems this program uses the characters that plot the best.]

Another creative student, Johnny LaPrarie, decided to make an exercise game similar in nature to the *Simon* game. His name for it is *shape'up*. He did a lot of work designing and coding sprites for the various exercises he wanted to include, but found he had to limit himself to 15 images since the COMAL error messages were resident in memory. I think that you will like his little game.

[Now our error messages in memory don't have this restriction; maybe he can add more sprites for his next project.]

Stacy Sominski decided to use the joystick with a matching game for elementary school kids. Her idea is simple and straightforward, as it should be for the very young. Her program, *matching*, works on discriminating between two attributes: shape and color. She was not very confident about her ability to accomplish this task, but she went right to work, planned carefully, and finished her project a full week before anyone else in the class. The *computer whizzes* were speechless.

[The programs from Toledo Christian High School are on Today Disk #14 virtually unedited. Many of them are listed here. Some had to be omitted due to extensive use of control characters which appear as unreadable graphics characters on our laser printer. We are happy to see what students are doing in the classroom and encourage other schools to send in their programs.]

More ►

Patterns

```
//////////////////////////////////
//     this program draws four  //
// different designs. the first //
// screen uses triangles, the   //
// second screen uses squares,  //
// the third uses pentagons,    //
// and the fourth and last uses //
// hexagons.                    //
//////////////////////////////////
// written by kevin page        //
//              11th grade      //
//              toledo christian //
//              toledo, ohio    //
//         on november 12, 1985 //
//////////////////////////////////
// delete "0:patterns-kp"       //
//                              //
// save "0:patterns-kp"         //
//////////////////////////////////
setup
tri'screen
wait
sqa'screen
wait
pent'screen
wait
hex'screen
wait
clear
settext
//////////////////////////////////
//          tri'screen          //
//////////////////////////////////
proc tri'screen
 moveto 100,50
 length:=100
 direc:=90
 triangle
 length:=20
 for loop1:=5 to 1 step -1 do
  moveto 100+x,50+y
  for loop:=1 to loop1 do
   triangle
   forward 20
  endfor loop
  x:=x+10
  y:=y+18
 endfor loop1
endproc tri'screen
//////////////////////////////////
//          sqa'screen          //
//////////////////////////////////
proc sqa'screen
 set
 length:=60
 for loop:=1 to 36 do
  square
  direc:=direc+10
  if loop<17 then
   length:=length-2
  else
   length:=length+2
  endif
 endfor loop
endproc sqa'screen
//////////////////////////////////
//          pent'screen         //
//////////////////////////////////
proc pent'screen
 set
 length:=40
 for loop:=1 to 5 do
  pentagon
  direc:=direc+72
 endfor loop
endproc pent'screen
//////////////////////////////////
//          hex'screen          //
//////////////////////////////////
proc hex'screen
 print chr$(147)
 setgraphic 1
 hideturtle
 clear
 home
 pencolor 6
 direc:=0
 length:=40
 print chr$(147)
 for loop:=1 to 6 do
  hexagon
  direc:=direc+60
 endfor loop
 for filloop:=1 to 12 do
  read c,filx,fily
  pencolor c
  fill filx,fily
 endfor filloop
endproc hex'screen
//////////////////////////////////
//            setup             //
//////////////////////////////////
proc setup
 background 12
 border 0
 pencolor 1
 x:=0
 y:=0
 dim question$ of 1
 setgraphic 0
 hideturtle
endproc setup
//////////////////////////////////
//          triangle            //
//////////////////////////////////
proc triangle
 setheading direc
 for tri:=1 to 3 do
  forward length
  left 120
 endfor tri
endproc triangle
print chr$(147)
//////////////////////////////////
//           square             //
//////////////////////////////////
proc square
 setheading direc
 for sqa:=1 to 4 do
  forward length
  left 90
 endfor sqa
endproc square
//////////////////////////////////
//          pentagon            //
//////////////////////////////////
proc pentagon
 setheading direc
 for pent:=1 to 5 do
  forward length
  left 72
 endfor pent
endproc pentagon
//////////////////////////////////
//          hexagon             //
//////////////////////////////////
proc hexagon
 setheading direc
 for hex:=1 to 6 do
  forward length
  left 60
 endfor hex
endproc hexagon
proc wait
 pencolor 6
 plottext 60,5,"press space bar to continue"
 repeat
 until key$=" "
endproc wait
proc set
 clear
 home
 pencolor 1
 direc:=0
endproc set
//////////////////////////////////
data 2,160,30,7,120,80,6,150,80
data 7,170,80,6,210,80,2,100,100
data 2,170,100,6,120,120,7,140,120
data 6,170,120,7,200,120,2,160,150
```

Psychadelic fungus

```
////////////////////////////////
//   the   wonderful   world   //
//                             //
//           of                //
//                             //
//     s p r i t e s           //
////////////////////////////////
//      by johnny laprarie     //
//         toledo christian    //
//         toledo, ohio        //
//         january 1986        //
////////////////////////////////
dim sprite$ of 64
for eddie:=1 to 3 do
 mulcommands
 define eddie,sprite$
 sprite$:=""
endfor eddie
for amp:=1 to 1 do
 hicommands
 define amp+3,sprite$
 sprite$:=""
endfor amp
setgraphic 0
hideturtle
background 1
for b:=1 to 100 do
 border rnd(1,13)
 plottext 80,100,"laprarie  productions"
 plottext 105,80,"*presents*"
endfor b
clear
left 145
pencolor 0
moveto 0,80
drawto 319,80
moveto 0,30
drawto 319,30
spritecolor 1,0
spritecolor 4,0
spritecolor 5,0
spriteback 10,14
spritesize 1,true,true
spritepos 1,140,80
spritesize 2,true,true
spritepos 2,200,100
spritesize 3,true,true
spritepos 3,80,100
//////////////
// animation //
///////////////
for m:=1 to 30 do
 pencolor rnd(2,15)
 plottext 80,150,"the psychadelic fungus"
 identify 5,4
 identify 4,4
 spritesize 5,true,true
 spritesize 4,true,true
 spritepos 4,5,100
 spritepos 5,250,100
 for sl:=1 to 3 do
  identify 1,(sl+1) mod 3+1
  identify 2,sl
  identify 3,sl
  for y:=1 to 100 do null
 endfor sl
endfor m
print chr$(147)
settext
////////////////////////////////
// proc multi color  commands //
////////////////////////////////
proc mulcommands
 for x:=1 to 63 do
  read d
  sprite$:=sprite$+chr$(d)
 endfor x
 sprite$:=sprite$+chr$(1)
endproc mulcommands
////////////////////////////////
// proc    hi rez    commands //
////////////////////////////////
proc hicommands
 for x:=1 to 63 do
  read d
  sprite$:=sprite$+chr$(d)
 endfor x
 sprite$:=sprite$+chr$(0)
endproc hicommands
/////////////////////
//  Eddie    # 1   //
/////////////////////
data 0,40,0
data 0,170,0
data 0,130,0
data 0,130,0
data 0,130,10
data 1,150,100
data 5,85,132
data 5,86,68
data 20,89,80
data 20,165,80
data 5,170,64
data 9,41,64
data 170,37,0
data 42,175,0
data 10,143,0
data 10,207,0
data 2,207,0
data 3,207,0
data 3,207,0
data 3,207,0
data 10,138,128
/////////////////////
//  Eddie    # 2   //
/////////////////////
data 0,40,0
data 0,170,0
data 0,130,0
data 0,130,0
data 0,130,10
data 1,150,100
data 5,85,132
data 21,86,68
data 20,89,80
data 4,165,80
data 4,170,64
data 5,169,64
data 168,165,0
data 40,175,0
data 10,143,0
data 10,207,0
data 2,207,0
data 3,207,0
data 3,207,0
data 3,207,0
data 10,138,128
/////////////////////
//  Eddie    # 3   //
/////////////////////
data 0,40,0
data 0,170,0
data 0,130,0
data 0,130,0
data 0,130,10
data 1,150,100
data 5,85,132
data 5,86,68
data 4,89,80
data 4,165,80
data 4,170,64
data 6,169,64
data 166,165,0
data 34,175,0
data 2,143,0
data 10,207,0
data 2,207,0
data 3,207,0
data 3,207,0
data 3,207,0
data 10,138,128
//////////////////
//  amp    # 1  //
//////////////////
data 0,0,0
data 0,0,0
data 0,0,0
data 3,255,192
data 3,128,64
data 3,86,64
data 3,128,64
data 3,255,64
data 3,0,64
data 3,60,64
data 3,231,64
data 3,211,64
data 3,203,64
data 3,60,64
data 3,0,64
data 3,255,192
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
```

Bill paint

```
// bill howard -- comal assignment
// march 17, 1986
//
// toledo christian schools
//
// "bill paint" drawing program
//
// initializing variables
//
firstx:=0
firsty:=0
firstb:=0
lbutton:=0
xrange:=0
yrange:=0
radius:=0
x:=0
y:=0
offx:=0
offy:=0
color:=2
mode:=0
//
// screen setup
//
print chr$(147)
instructions
print "push space bar to start"
repeat
until key$=" "
menu
dim sprite$ of 64
icons
frame 46,319,17,199
//
// beginning of main program
//
repeat
 koala(2,xloc,yloc,lb,rb)
 if xloc<=7 or yloc>=249 then
  null
 else
  scale
 endif
 //
 // checking for which mode to use
 //
 if x<45 and lb=1 then
  if y>166 then
   mode:=1
   offx:=5
   offy:=18
   moveto x,y
   hidesprite 5
  elif y>132 then
   mode:=2
   offx:=18
   offy:=18
   hidesprite 5
  elif y>98 then
   mode:=3
   offx:=10
   offy:=10
   identify 3,3
  elif y>64 then
   mode:=4
   offx:=11
   offy:=9
   identify 3,3
  elif y>30 then
   mode:=5
   offx:=11
   offy:=9
   identify 3,3
  elif y>0 then
   mode:=6
  endif
 else
  spritepos 3,x-offx,y+offy
 endif
 //
 // call procedure according to mode
 //
 if mode=1 then
  draw
 elif mode=2 then
  fillin
 elif mode=3 then
  line
 elif mode=4 then
  box
 elif mode=5 then
  circle
 elif mode=6 then
  newscreen
 endif
 //
 // color change
 //
 if x>=55 and y<17 and lb=1 then
  color:=int((x-55)/16)+2
  if color>15 then
   if x>285 and x<301 then
    color:=1
   elif x>301 then
    color:=0
   endif
  endif
  pencolor color
  border color
 endif
until false
//
// end of main program
// ****************
// icon procedure
// ****************
// read and define 5 sprites
//
proc icons
 for mainloop:=1 to 5 do
  for loop:=1 to 63 do
   read d
   sprite$:=sprite$+chr$(d)
  endfor loop
  sprite$:=sprite$+chr$(0)
  define mainloop,sprite$
  spritesize mainloop,false,false
  sprite$:=""
 endfor mainloop
 spritecolor 1,0
 spritecolor 2,0
 spritecolor 3,0
 spritecolor 4,0
 spritecolor 5,0
 identify 1,1
 identify 2,2
 identify 3,5
 // pencil
 spritepos 1,10,195
 //paint can
 spritepos 2,9,163
 // line and box
 //
 moveto 7,110
 drawto 34,120
 moveto 12,70
 for x:=1 to 4 do
  forward 20
  right 90
 endfor x
 // circle and new
 //
 circle'as(21,45,10)
 identify 4,4
 spritepos 4,10,25
endproc icons
//
// ****************
// menu procedure
// ****************
proc menu
 border 0
 background 1
 pencolor 0
 setgraphic 1
 hideturtle
 //
 // vertical menu
 //
 moveto 45,0
 drawto 45,199
 moveto 0,0
 drawto 0,199
 for x:=199 to 0 step -34 do
  moveto 0,x
  drawto 45,x
 endfor x
 //
 // horizontal color bar
 //
 moveto 55,0
 drawto 55,17
 moveto 55,17
 drawto 311,17
 for x:=55 to 311 step 16 do
  moveto x,0
  drawto x,17
 endfor x
```

More ►

```
 pencolor color
 for j:=55 to 266 step 16 do
  fill j+5,0
  color:=(color+1) mod 16
  pencolor color
 endfor j
 pencolor 0
 fill 297,3
endproc menu
// ************************
// circle aspect procedure
// ************************
proc circle'as(x,y,r) closed
 y':=0
 penup
 for i:=1 to 64 do
  t:=r*.995004165-y'*.0998334166
  y':=y'*.995004165+r*.0998334166
  r:=t
  sx:=1.4*r+x
  sy:=y-y'
  drawto sx,sy
  pendown
 endfor i
endproc circle'as
// ****************
// koala procedure
// ****************
proc koala(p,ref x,ref y,ref lb,
 ref rb) closed // wrap line
 j:=15-peek(3-p+56319) mod 16
 lb:=false; rb:=false
 if j=4 then lb:=true
 if j=8 then rb:=true
 c:=56320
 s:=54272
 poke c+13,1
 d:=peek(c+2)
 poke c+2,192
 poke c,64*p
 x:=peek(s+25); y:=255-peek(s+26)
 poke c+2,d
 poke c+13,129
endproc koala
// ****************
// scale procedure
// ****************
proc scale
 x:=(xloc*320)/251
 y:=(yloc*200)/250
endproc scale
// ******************
// drawing procedure
// ******************
proc draw
 identify 3,1
 if lb=1 then
  spritepos 3,x-offx,y+offy
  drawto x,y
 else
  identify 3,1
  spritepos 3,x-offx,y+offy
  moveto x,y
 endif
endproc draw
// *****************
// fillin procedure
// *****************
proc fillin
 identify 3,2
 if lb=1 then
  spritepos 3,x-offx,y+offy
  fill x,y
 else
  spritepos 3,x-offx,y+offy
 endif
endproc fillin
// ***************
// line procedure
// ***************
proc line
 if lb=1 and x>45 and y>17 then
  identify 5,3
  spritepos 5,x-offx,y+offy
  firstx:=x
  firsty:=y
  firstb:=1
 endif
 if rb=1 and firstb=1 then
  moveto firstx,firsty
  drawto x,y
 endif
endproc line
// ********************
// newscreen procedure
// ********************
proc newscreen
 if lb=1 then lbutton:=1
 if x>45 and y>17 then
  lbutton:=0
 elif rb=1 and lbutton=1 then
  color:=2
  hidesprite 3
  hidesprite 5
  frame 0,319,0,199
  menu
  restore
  icons
  frame 46,319,17,199
  mode:=0
 endif
endproc newscreen
// ****
// box
// ****
proc box
 if lb=1 and x>45 and y>17 then
  identify 5,3
  spritepos 5,x-offx,y+offy
  firstx:=x
  firsty:=y
  firstb:=1
 endif
 if rb=1 and firstb=1 then
  moveto firstx,firsty
  drawto x,firsty
  moveto x,firsty
  drawto x,y
  moveto x,y
  drawto firstx,y
  moveto firstx,y
  drawto firstx,firsty
  moveto firstx,firsty
 endif
endproc box
// *****************
// circle procedure
// *****************
proc circle
 if lb=1 and x>45 and y>17 then
  identify 5,3
  spritepos 5,x-offx,y+offy
  firstx:=x
  firsty:=y
  firstb:=1
 endif
 xrange:=abs(firstx-x)
 yrange:=abs(firsty-y)
 radius:=sqr(xrange^+yrange^)
 if rb=1 and firstb=1 then
  circle'as(firstx,firsty,radius)
 endif
endproc circle
// pencil
data 0,0,0,0,0,0,0,15,0,0,8,128
data 0,16,128,0,25,0,0,39,0,34,0
data 0,66,0,0,68,0,0,132,0,0,136,0
data 1,8,0,1,16,0,1,224,0,1,192,0
data 1,128,0,1,0,0,0,0,0,0,0,0,0,0
// paint can
data 0,0,0,0,56,0,0,68,0,0,68,0
data 0,76,0,0,86,0,0,101,0,0,68,192
data 0,132,112,1,10,56,2,4,56,4,0,120
data 4,0,184,2,1,56,1,2,56,0,132,56
data 0,72,48,0,48,32,0,0,0,0,0,0,0,0,0
// cross hairs
data 0,0,0,0,0,0,0,0,0,0,0,0
data 0,24,0,0,24,0,0,24,0,0,24,0
data 1,255,128,1,255,128,0,24,0,0,24,0
data 0,24,0,0,24,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0,0,0,0,0,0,0
//
//trash can
//
data 0,0,0,0,0,0,0,0,0,0,255,0
data 3,0,192,15,255,240,0,0,0,3,255,192
data 2,66,64,2,66,64,2,66,64,2,66,64
data 2,66,64,2,66,64,2,66,64,3,66,64
data 2,165,64,1,255,128,0,0,0,0,0,0,0,0,0
data 0,0,0,0,0
//
//arrow
//
data 0,0,0,0,252,0,0,248,0,0,240,0
data 0,248,0,0,204,0,0,134,0,0,3,0
data 0,1,128,0,0,192,0,0,0,0,0,0
data 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
data 0,0,0,0,0,0
```

More ►

```
// ************************
// instructions procedure
// ************************
proc instructions
 background 1
 border 0
 pencolor 0
 print chr$(13),chr$(13),chr$(13)
 print tab(16),"bill paint"
 print chr$(13),chr$(13)
 print "to use line, circle and"
 print "box commands, push left"
 print "button for first point"
 print "then the right button"
 print "to execute the command."
 print "to use clear command,"
 print "position the cursor in"
 print "the trash can box and"
 print "press the left button &"
 print "then the right. this"
 print "clears the whole screen"
endproc instructions ■
```

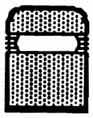
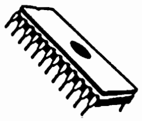
Helicopter

```
// joe postma
// toledo christian
// toledo, ohio
// delete "0:multi-heli-jp"
// date 1/7/86
// save "0:multi-heli-jp"
dim copter$ of 64
background 0
border 0
pencolor 1
si:=1
for loop:=1 to 63 do
 read d
 copter$:=copter$+chr$(d)
endfor loop
copter$:=copter$+chr$(1)
define 1,copter$
copter$:=""
for x:=1 to 63 do
 read d
 copter$:=copter$+chr$(d)
endfor x
copter$:=copter$+chr$(1)
define 2,copter$
copter$:=""
for x:=1 to 63 do
 read d
 copter$:=copter$+chr$(d)
endfor x
copter$:=copter$+chr$(1)
define 3,copter$
copter$:=""
for x:=1 to 63 do
 read d
 copter$:=copter$+chr$(d)
endfor x
copter$:=copter$+chr$(0)
define 4,copter$
copter$:=""
setgraphic 0
hideturtle
spritecolor 1,2
spritecolor 2,15
spritecolor 3,7
spriteback 10,14
spritesize 1,true,true
spritesize 2,true,true
for x:=1 to 125 do
 identify 1,1
 spritepos 1,80,x
 for y:=1 to 10 do null
endfor x
for x:=1 to 500 do
endfor x
hidesprite 1
for x:=80 to 330 do
 spritepos 1,x,125
 identify 1,3
 for y:=1 to 10 do null
endfor x
for x:=1 to 500 do
endfor x
hidesprite 1
for x:=0 to 300 do
 spritepos 1,x,2*x
 identify 1,4
endfor x
for x:=0 to 330 do
 r:=rnd(100,102)
 identify 1,4
 spritepos 1,x,r
endfor x
settext
//copter #1
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 2,255,240
data 0,12,0
data 176,170,128
data 42,168,0
data 10,170,40
data 0,170,160
data 0,63,192
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
//copter #2
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 2,255,240
data 0,12,0
data 176,170,128
data 42,168,0
data 10,170,40
data 10,170,160
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
// copter #3
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 2,255,240
data 0,12,0
data 176,170,128
data 42,168,0
data 10,170,40
data 1,106,160
data 0,63,192
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
//hires copter
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 3,255,224
data 0,8,0
data 32,127,128
data 63,242,0
data 7,242,96
data 0,127,192
data 0,4,0
data 0,63,128
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0 ■
```

Sprite motion

```
// sprite motion
// michele bostdorff
// toledo christian
// toledo, ohio
// january 1986
//
setgraphic 0
background 0
clear
dim s$ of 64
for x:=1 to 63 do
 read d
 s$:=s$+chr$(d)
endfor x
s$:=s$+chr$(1)
define 1,s$
s$:=""
setgraphic 0
hideturtle
spritecolor 1,6
spriteback 2,4
spritesize 1,true,true
identify 1,1
repeat
 for x:=1 to 250 do
  spritepos 1,x,150
 endfor x
 for x:=250 to 1 step -1 do
  spritepos 1,x,150
 endfor x
until false
data 0,0,0
data 0,0,0
data 16,18,163
data 16,16,131
data 16,16,131
data 16,16,131
data 16,16,131
data 16,16,131
data 21,80,131
data 16,16,131
data 16,16,131
data 16,16,128
data 16,16,128
data 16,16,131
data 16,18,163
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0
data 0,0,0 ■
```

Keywords, Procedures, Packages

by Richard Bain

COMAL 2.0 has a wide means to determine what you intend to do every time you type something on the keyboard and hit *<return>*. Maybe you typed in a keyword such as **RUN** or **LIST**. Perhaps you called one of your program's procedures from the immediate mode. You may have used a command from a package. These possibilities can be confusing because you only typed in a word or two. However, to get these different commands to work, you must know how to set them up.

A keyword is a command that COMAL always knows about. COMAL will always know what you mean if you type in **EDIT** or **PRINT** or any of the other words you see in uppercase when you list a COMAL 2.0 program. You don't have to do anything special to make these commands work.

Procedures and functions are like keywords, but you must define them first. They only last as long as the the program they are from is unchanged in memory. For example, COMAL 2.0 has the built in function **PI**, but COMAL 0.14 does not. COMAL 0.14 can use a function to simulate this keyword:

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

You must **RUN** the program before COMAL 0.14 will know about **pi**. Once the program is **RUN**, you can issue the following command in direct mode:

```
PRINT pi
```

COMAL 2.0 already knows about **PI**, but can learn other procedures and functions from the **SCAN** command as well as **RUN**.

Packages fall somewhere between keywords and procedures. They are like keywords in that you will never see their definition listed in a program, only the calls to them. Some packages such as **font**, **sprites**, and **sound** are built into the COMAL cartridge. Additional packages may be disk loaded or on EPROM (like Super Chip). However, these packages are different from keywords because they still must be learned by COMAL before they can be used. They are not learned with the **SCAN** or **RUN** commands the way procedures are, but with the **USE** command. **USE** *<packagename>* must be typed before any of the package's procedures are called in the immediate mode. The **USE** *<packagename>* command must also appear in any program or closed procedure before its package commands can be used.

Packages are also like procedures or functions. In fact, procedure **scroll'down** from the **system2** package in Super Chip and procedure **scroll'down** from *COMAL Today #8* do exactly the same thing. The only difference is that before calling the one from Super Chip, you must enter the command: **USE system2**:

```
USE system2
scroll'down
```

Before calling the one that is a procedure, you need a **RUN**, (or **SCAN** in COMAL 2.0):

```
SCAN
scroll'down
```

Without the prior **USE** or **SCAN**, you could get the error:

unknown statement or procedure

More ►

Packages come in three different forms. There is the ROM or EPROM form. These are inside your COMAL cartridge. You and your programs can use them any time you turn your computer on. The built in packages such as <u>sprites</u> come in this form. So do the packages of Super Chip. The other two kinds of packages are disk loaded. They are ROMMED packages and RAM packages. These packages must be linked into COMAL before they can be used. The link command reads the package from disk into RAM memory:

```
LINK "pkg.cmon"
USE cmon
```

ROMMED packages such as <u>cmon</u> from *COMAL Today #10* will remain in memory until the computer is turned off, but must be relinked the next time the power is turned back on. They add procedures which can be called from the immediate mode, or from programs, after the **USE** command is issued.

RAM packages are only in memory temporarily. They are lost as soon as you type **DISCARD**, **NEW**, **LOAD**, or **ENTER**. (The program in memory may be edited, but then **USE** must be reentered.) However, they have one advantage over ROMMED packages. RAM packages are saved together with the programs they are used with. This means you can reload the program at a later time and the package will already be linked to it. A ROMMED package would need to be linked in separately.

If keywords, normal procedures, and package procedures are so alike, why do you have to enter **RUN** before calling one or enter **USE** before calling another? The answer lies in how COMAL determines what you are asking it. If you issue a command from the immediate mode, COMAL will first check to see if it is a keyword. If it is, COMAL will execute it. If not, COMAL will check its name table to see if it can find the procedure there. Unfortunately, when you turn the computer on, the name table is empty. COMAL must learn the new names.

One way COMAL learns new procedures is with the **USE** command. When you enter USE <*packagename*>, you may not *see* anything happen, but COMAL adds all the package's procedures and functions to the name table so they may be called later. To see this better, type:

```
NEW
SIZE
USE system
SIZE
```

Notice the reduction in free memory after the **USE** command. Some might prefer to have the built in package procedures available on power up before issuing the **USE** command. If they were, the name table would steal several thousand bytes from COMAL's program space for procedures and functions which might never be called.

Having COMAL learn a new procedure by typing or merging it in is a little more complicated. The name will go into the name table right away, but COMAL still will not know it until you type in **RUN** or **SCAN**. There are two reasons for this. The first is that even though COMAL puts the name in the name table, it does not include the fact that it is a procedure. COMAL won't recognize the name as a procedure when it sees it. The second reason is that COMAL wants to know that the procedure is *correct* before executing it. This is exactly what **SCAN** does and explains why you can't call a procedure immediately after you alter the program it came from. ▣

Calendars

Eric Haas and Kevin Quiggle sent us calendar programs. Either program could be used to make memo pads, or simply as a means to see if your sixteenth birthday falls on a schoolday. Eric's program is in COMAL 0.14 and will print to any standard printer. Kevin's COMAL 2.0 program, listed below, prints to the 1520 plotter. Both programs are on *Today Disk #14*.

Further References:

The Real Julian Day, COMAL Today #9, page 76
1520 Plotter Driver Routines, COMAL Today #7, page 62

```
instructions
get'dates
init
TRAP
  p'color(color)
  FOR month:=first'month TO last'month DO
    p'set'absolute
    plot'calendar(month,year)
  ENDFOR month
HANDLER
  SELECT OUTPUT "ds:"
  PAGE
  PRINT "ERROR:";ERRTEXT$
  CLOSE
  STOP
ENDTRAP
SELECT OUTPUT "ds:"
END "End of program"
//
PROC init
  DIM month'name$(12) OF 9
  DIM month'length#(12)
  DIM weekday$(7) OF 3
  DIM title$ OF 18
  read'months
  read'week'days
  size:=4
ENDPROC init
//
PROC read'months
  RESTORE months
  FOR month:=1 TO 12 DO
    READ month'name$(month),month'length#(month)
  ENDFOR month
ENDPROC read'months
//
PROC read'week'days
  RESTORE weekdays
  FOR day:=1 TO 7 DO
    READ weekday$(day)
  ENDFOR day
ENDPROC read'week'days
//
PROC instructions
  PAGE
  PRINT AT 1,5: "Calendar Plotter by Kevin Quiggle"
  PRINT AT 3,8: "(1520 plotter required)"
  PRINT AT 5,1: "This program prints a calendar (or part",
  PRINT AT 6,1: "of a calendar) using a 1520 plotter."
  PRINT AT 8,1: "You must enter the year, first month,"
  PRINT AT 9,1: "and last month for the calendar.  You"
  PRINT AT 10,1: "may also select the pencolor for the"
  PRINT AT 11,1: "calendar."
ENDPROC instructions
//
PROC get'dates
  INPUT AT 16,1: "Enter the year (YYYY format): ": year
  INPUT AT 17,1: "Enter the first month (1-12): ": first'month
  INPUT AT 18,1: "Enter the last month (1-12):  ": last'month
  INPUT AT 19,1: "Enter pen color (1=blue,2=green,3=red,
  4=black): ": color // wrap line
ENDPROC get'dates
//
PROC plot'calendar(month,year)
  p'home
  plot'box(0,0,119.75,119.75)
  plot'box(4,20,115.75,115.75)
  plot'grid
  plot'title(month,year)
  plot'day'names
  plot'day'numbers(month,year)
  p'moveto(0,-size*150)
ENDPROC plot'calendar
//
PROC plot'day'numbers(month,year)
  IF month=2 AND THEN leap'year(year) THEN month'len
  gth#(2):=29 // wrap line
  week'day:=day'of'week(month,1,year)
  week:=0
  FOR day:=1 TO month'length#(month) DO
    p'moveto(size*(6+15.75*week'day),-size*(30.75+15*week))
    p'char(STR$(day))
    week'day:+1
    IF week'day=7 THEN
      week'day:=0
      week:+1
    ENDIF
  ENDFOR day
ENDPROC plot'day'numbers
//
PROC plot'title(month,year)
  p'charsize(size-2)
  title$:=month'name$(month)+SPC$(2)+STR$(year)
  offset:=(20-LEN(title$))/2*6
  p'moveto(offset*size,-15*size)
  p'char(title$)
ENDPROC plot'title
//
PROC plot'day'names
  p'charsize(size-3)
```

More ►

```
 FOR day:=0 TO 6 DO
  p'moveto((8+15.75*day)*size,-25*size)
  p'char(weekday$(day+1))
 ENDFOR day
ENDPROC plot'day'names
//
PROC plot'box(x1,y1,x2,y2)
 scale(x1,y1,x2,y2)
 p'moveto(x1,y1)
 p'drawto(x2,y1)
 p'drawto(x2,y2)
 p'drawto(x1,y2)
 p'drawto(x1,y1)
ENDPROC plot'box
//
PROC plot'grid
 FOR vertical:=1 TO 6 DO
  connect(4+15.75*vertical,20,4+15.75*vertical,115.75)
 ENDFOR vertical
 FOR horizontal:=0 TO 5 DO
  connect(4,25.75+15*horizontal,115.75,25.75+15*horizontal)
 ENDFOR horizontal
ENDPROC plot'grid
//
PROC connect(x1,y1,x2,y2)
 scale(x1,y1,x2,y2)
 p'moveto(x1,y1)
 p'drawto(x2,y2)
ENDPROC connect
//
PROC scale(REF x1,REF y1,REF x2,REF y2)
 x1:=size*x1
 x2:=size*x2
 y1:=-size*y1
 y2:=-size*y2
ENDPROC scale
//
FUNC leap'year(year) CLOSED
 IF year MOD 100=0 THEN
  IF year MOD 400=0 THEN
   RETURN (TRUE)
  ELSE
   RETURN (FALSE)
  ENDIF
 ELSE
  IF year MOD 4=0 THEN
   RETURN (TRUE)
  ELSE
   RETURN (FALSE)
  ENDIF
 ENDIF
ENDFUNC leap'year
//
FUNC day'of'week(month,day,year) CLOSED
 IF month<=2 THEN
  month:+12
  year:-1
 ENDIF
 x:=day+2*month+INT(.6*(month+1))
 x:+year+INT(year/4)-INT(year/100)+INT(year/400)+1
 RETURN (INT((x MOD 7)+.5))
ENDFUNC day'of'week
//
FUNC day'of'year(month,day,year) CLOSED
 doy:=day+(month-1)*30+INT((month+1)*.61)-2
 IF month<=2 THEN
  doy:+month
 ELIF NOT leap'year(year) THEN
  doy:-1
 ENDIF
 RETURN (doy)
ENDFUNC day'of'year
//
months:
DATA "JANUARY",31
DATA "FEBRUARY",28
DATA "MARCH",31
DATA "APRIL",30
DATA "MAY",31
DATA "JUNE",30
DATA "JULY",31
DATA "AUGUST",31
DATA "SEPTEMBER",30
DATA "OCTOBER",31
DATA "NOVEMBER",30
DATA "DECEMBER",31
weekdays:
DATA "SUN"
DATA "MON"
DATA "TUE"
DATA "WED"
DATA "THU"
DATA "FRI"
DATA "SAT"
//
//save "1520driver
// by kevin quiggle
//
PROC p'open(sa) CLOSED
 SELECT OUTPUT "u6:/s"+STR$(sa)
ENDPROC p'open
//
PROC p'close CLOSED
 SELECT OUTPUT "ds:"
ENDPROC p'close
//
PROC p'char(c$)
 p'open(6)
 PRINT 1
 p'open(0)
 IF c$<>"" THEN
  PRINT c$,
 ELSE
  PRINT c$
 ENDIF
 p'close
ENDPROC p'char
//
PROC p'home
 p'open(1)
 PRINT "h"
 p'close
ENDPROC p'home
//
PROC p'init
 p'open(1)
```

More ►

Calendars - continued

```
  PRINT "i"
  p'close
ENDPROC p'init
//
PROC p'moveto(x,y)
  p'open(1)
  PRINT "m";x;CHR$(29);y;CHR$(29)
  p'close
ENDPROC p'moveto
//
PROC p'drawto(x,y)
  p'open(1)
  PRINT "d";x;CHR$(29);y;CHR$(29)
  p'close
ENDPROC p'drawto
//
PROC p'move(x,y)
  p'open(1)
  PRINT "r";x;CHR$(29);y;CHR$(29)
  p'close
ENDPROC p'move
//
PROC p'draw(x,y)
  p'open(1)
  PRINT "j";x;CHR$(29);y;CHR$(29)
  p'close
ENDPROC p'draw
//
PROC p'reset
  p'open(7)
  PRINT
  p'close
ENDPROC p'reset
//
PROC p'color(color)
  p'open(2)
  PRINT color
  p'close
ENDPROC p'color
//
PROC p'charsize(size)
  p'open(3)
  PRINT size
  p'close
ENDPROC p'charsize
//
PROC p'rotchar(rot)
  p'open(4)
  PRINT rot
  p'close
ENDPROC p'rotchar
//
PROC p'scribe(brk)
  p'open(5)
  PRINT brk
  p'close
ENDPROC p'scribe
//
PROC p'set'absolute
  p'open(0)
  PRINT CHR$(13)
  p'close
ENDPROC p'set'absolute
```

JULIAN DAY REVISITED

Tom Kuiper's article, *"The Real Julian Day"* on page 76 of *COMAL Today #9* is quite correct. *COMAL Today* readers should be made aware however, that numerous computer systems implement a "julian date" function beginning on January 1 of the current year. I have no way of knowing for certain, but I suspect that the "incorrect" julian day is probably the more common usage amoung computer systems. - Kevin Quiggle, Detroit, MI

In the State of Wisconsin computing center, a julian date is a two digit year followed by three digits specifying how many days into the current year the day is. February 1, 1986 would be 86.032. Note that a period is used to separate the year from the day number.

Directory Sort

by Jack Baldridge

A program I have used quite a bit since before I ever heard of COMAL is one which sorts the directory on a disk. I find it much easier to search for a file on a disk which has them in alphabetical order. After I learned about COMAL, however, it irritated me to go into BASIC when I wanted to sort a disk directory.

Finally, I developed a COMAL 2.0 program to sort a 1541 disk directory. When I compared the time needed to sort a disk directory using this program with that of the BASIC program, I found a big difference. Sorting identical disks, the COMAL program ran about four times faster. I really knew nothing about how efficient the BASIC program might be, since I hadn't referred to it while writing the COMAL program. Next I adapted my program to run in COMAL 0.14.

I found the run times for the three programs for identical disks consistent but rather surprising. The BASIC program ran in about 48 seconds, the COMAL 0.14 program in about 40 seconds, while the COMAL 2.0 version took only a little over 12 seconds. It looks to me as though that cartridge has some pretty powerful routines built into it. Both the 0.14 and 2.0 versions of the program are on *Today Disk #14.*

Further Reference:

Fast Dir Revisited, COMAL Today #13, page 50
Sorting It Out, COMAL Today #10, page 18

Custom Directories

by Jim Kaminski

Many times I have needed a listing of the file directory on a disk in a different format than the standard one you get with:

```
SELECT "LP:"
CAT
```

In this case, my need was for one which was printed very tiny and in two columns (so it would fit on a disk label). I found a modified BASIC listing which almost did it, but when trying to change it I realized how poorly it was programmed. I could not easily change 'just a few lines' to make it work as I wanted.

At last I gave up and wrote a 'generic' directory lister program. COMAL was the language of choice because of easy modularity, and good interactive debugging features. The listing is in COMAL 2.0, but to make it work in COMAL 0.14 a number of changes are needed which I did not have the time to implement (mainly in the OPEN, SPC$, MOUNT, DIR, and GET$ statements). This I leave as a task for another.

This program, as currently written, will print a very tiny two-column listing of a disk directory. It can be easily modified to produce one, three, or more columns; to print in expanded or italic letters; to have a different line spacing; and even have different spacings between columns or parts of a directory entry (such as between the filename and the filetype). Note that this program will read in all entries into an array and then print out the columns such that the up and down order is maintained in each column.

In this 'generic' program, no effort was made to enhance speed or to allow for interactive changing of parameters. These items could be added, but I felt that the features of COMAL 2.0 allowed me to write the program in a relatively simple way, yet change it easily later on as my needs changed.

The program is written using variables throughout so that changes are implemented by revising the values in only one module.

COL#: number of columns
OFFSET#: Blank spaces at start of line
DIVIDER#: Blank spaces between columns
S#: Space between parts on the line
B#: Maximum digits in filesize blocks

Of course, the device numbers, secondary addresses, and printer specific codes can all be easily changed to customize the program for a particular system (which is why some of the existing public domain programs are not good enough for some applications). Upper and lower case ASCII/PETSCII translations can be implemented by using the proper secondary address for the printer/interface in the system.

```
   "CARTRIDGE DEMO 3" D3 2A
23 "ALL'AT'ONCE2"      PRG    1 "BAT.COMMANDS"      SEQ
 1 "--DEMO PROGRAMS-"  SEQ    1 "BAT.FONT'CMDS"     SEQ
17 "1520 PLOTTER"      PRG    1 "---DATA FILES---"  SEQ
14 "ARABESQUE2"        PRG   19 "DAT.BWV783"        SEQ
13 "ARABESQUE3"        PRG   15 "DAT.BWV784"        SEQ
11 "BATCHFILE'EDITOR"  PRG   17 "DAT.BWV801"        SEQ
18 "BINARY'COUNTER"    PRG    1 "---FONT FILES---"  SEQ
 9 "CHECK'CARTRIDGE"   PRG   17 "FONT.COMPUTER"     SEQ
 9 "CURVE3"            PRG   17 "FONT.D&D"          SEQ
 9 "CURVE4"            PRG   17 "FONT.GREEK"        SEQ
16 "EXTEND'COLOR"      PRG   17 "FONT.HEBREW"       SEQ
16 "FILE'CARD'MAKER"   PRG   17 "FONT.ROOSKI"       SEQ
13 "GRAPH4"            PRG   17 "FONT.STANDARD"     SEQ
12 "GRAPH5"            PRG    1 "-----PICTURES----" SEQ
13 "GRAPH6"            PRG   40 "HRG.NORTHWEST"     SEQ
18 "KOALA'TO'2.0"      PRG   36 "HRG.WORLD'MAP"     SEQ
 3 "PICTURE'LOADER"    PRG    1 "--PROCS & FUNCS-"  SEQ
15 "PLAYSCORE2"        PRG    3 "FUNC.MODEM'GET"    SEQ
17 "PROTECT64"         PRG    3 "PROC.PLOTTER"      SEQ
14 "READ'DIRECTORY"    PRG    4 "PROC.SHOW'SPRITE"  SEQ
13 "RUNNING'MEN"       PRG    1 "---SHAPE FILES--"  SEQ
 9 "SHOW'CHARACTER"    PRG    1 "SHAP.BAT0"         SEQ
16 "SHOWLIBS"          PRG    1 "SHAP.BAT1"         SEQ
32 "SIDMONITOR"        PRG    1 "SHAP.BAT2"         SEQ
 8 "SOUND'ENVELOPE"    PRG    1 "SHAP.MEN0"         SEQ
25 "SPRITE'EDITOR"     PRG    1 "SHAP.MEN1"         SEQ
18 "STAMPSPRITE"       PRG    1 "SHAP.MEN2"         SEQ
29 "VIEW'FONTS"        PRG    0 BLOCKS FREE
 1 "--BATCH FILES---"  SEQ
```

Multi Directory

by Ray Carter

This program was designed to be the ultimate Directory listing program. As set up it will read in directories from several disks (up to a total of 400 entries) and print them out on a GEMINI 10X printer in 80 lines of 5 columns each in 17 cpi format. Comments are included at the beginning of the program to explain how to modify it to other configurations. With slight modifications, it prints out 120 lines of 5 columns in 17 cpi superscript mode. The results are as expected, although the print is a little small. The comments should allow modification for other printers, as well.

Note that the program uses the technique for reading a directory which is described in my article *Fast Directory Revisited* on page 50 of *COMAL Today #12*. Another interesting feature is the **filter$** string function used to filter out non-standard characters (which wreak havoc with the Cardco +G interface) and permit the smaller character sets to work.

```
// multi-dir-400 by ray carter
// to modify the program --
// MAXPRT - max # char allowed per line
// NROWS - total number of lines to be printed per page
// PROC SETPRINTER to send appropriate escape sequences
// to set your printer up for proper character size and line feed
USE system
DIM a$ OF 1, junk$ OF 142, b$ OF 1
maxprt:=136
size:=24
ncols:=5
nrows:=80
DIM dirs$(nrows,ncols) OF size
initialize
filldir
setprinter
printdir
STOP
//
PROC initialize
  PAGE
  PRINT "This program prints directories from several disks in"
  PRINT "multi-column format. It is set up to print 80 lines of"
  PRINT "5 columns to a GEMINI-10X printer using 17"
  PRINT "characters per inch mode."
  PRINT "Please turn on the printer and position the paper."
  PRINT "Hit any key to continue!"
  b$:=inkey$
  PRINT "Insert the first disk, and press a key to continue."
  b$:=inkey$
  FOR i:=1 TO nrows DO
    FOR j:=1 TO ncols DO
      dirs$(i,j):=SPC$(24)
    ENDFOR j
  ENDFOR i
  OPEN FILE 5,"lp:",WRITE
ENDPROC initialize
//
PROC filldir
  i:=1; j:=1
  REPEAT
    readdir
    bumpindex
    IF a$<>"n" THEN
      INPUT "another disk [y/n]: ": a$
      PRINT
      PRINT "Insert the next disk, and press a key to continue"
      b$:=inkey$
      icols:=j
      irows:=i
    ELSE
      PRINT "The current disk won't be included in the"
      PRINT "listing--no room. You may want to run the"
      PRINT "program again and use it as the first disk. If you"
      PRINT "do run the program again, adjust the paper first."
    ENDIF
  UNTIL a$="n"
ENDPROC filldir
//
PROC bumpindex
  i:=i+1
  IF i>nrows THEN
    i:=1
    j:=j+1
  ENDIF
  IF j>ncols THEN a$:="n"
ENDPROC bumpindex
//
PROC printdir
  tabstop:=INT(maxprt/ncols)
  FOR i:=1 TO nrows DO
    FOR j:=1 TO ncols DO
      IF (i<irows OR j<icols) THEN
        PRINT FILE 5: TAB((j-1)*tabstop+1),dirs$(i,j),
      ENDIF
    ENDFOR j
    PRINT FILE 5: ""
  ENDFOR i
ENDPROC printdir
//
PROC readdir
  PASS "i0:"
```

More ►

```
  OPEN FILE 2,"u8:$0/s1/t+/d+",READ
  junk$:=GET$(2,142)
  dirs$(i,j):=filter$(GET$(2,23))
  PRINT dirs$(i,j)
  bumpindex
  IF a$="n" THEN
    CLOSE FILE 2
    RETURN
  ENDIF
  junk$:=GET$(2,89)
  k:=0
  REPEAT
    x$:=GET$(2,30)
    k:=k+1
    IF (k MOD 8)<>0 THEN junk$:=GET$(2,2)
    IF ORD(x$(1))<>0 THEN
      CASE ORD(x$(1)) BITAND $0f OF
      WHEN 0
        dirs$(i,j)(22:24):="del"
      WHEN 1
        dirs$(i,j)(22:24):="seq"
      WHEN 2
        dirs$(i,j)(22:24):="prg"
      WHEN 3
        dirs$(i,j)(22:24):="usr"
      WHEN 4
        dirs$(i,j)(22:24):="rel"
      OTHERWISE
        dirs$(i,j)(22:24):="unk"
      ENDCASE
      IF ORD(x$(1))<127 THEN dirs$(i,j)(21:21):="*"
      IF ORD(x$(1)) BITAND $c0=$c0 THEN dirs$(i,j)(21:21):=
      "<" // wrap line
      dirs$(i,j)(5:20):=filter$(x$(4:19))
      fsiz:=ORD(x$(29))+256*ORD(x$(30))
      dirs$(i,j)(1:3):=STR$(fsiz)
      bumpindex
    ENDIF
  UNTIL a$="n" OR EOF(2)
  CLOSE FILE 2
ENDPROC readdir
//
PROC setprinter
  OPEN FILE 27,"lp:/s4",WRITE
  PRINT FILE 27: CHR$(27)+CHR$(66)+CHR$(3), //set 17 cpi
  PRINT FILE 27: CHR$(27)+CHR$(48) //set to 8 lpi
  CLOSE FILE 27
ENDPROC setprinter
//
FUNC filter$(x$) CLOSED
  a$:=""
  FOR i:=1 TO LEN(x$) DO
    b:=ORD(x$(i:i))
    IF b<32 THEN b:=63
    IF (b>90 AND b<97) THEN b:=63
    IF b=160 THEN b:=32
    IF (b>122 AND b<193) THEN b:=63
    IF b>218 THEN b:=63
    a$:=a$+CHR$(b)
  ENDFOR i
  RETURN a$
ENDFUNC filter$
```

House

by Phyrne Bacon

When I demonstrated the use of **DRAWTO** and **MOVETO** in my lecture on COMAL graphics, I drew a house added one star (to demonstrate **PLOT**). I added "house" at the bottom of the picture (to demonstrate **PLOTTEXT**). One of my students asked me to add some random stars, so I added ten. Then I added a window using a procedure box to draw the window panes, and changed "house" to "house & window" in **PLOTTEXT**.

```
// delete "house'and'window"
// by Phyrne Bacon
// save  "house'and'window"
//
setgraphic 0
hideturtle
fullscreen
border 11
background 11
clear
pencolor 1
drawto 10,99
drawto 310,99
moveto 160,170
drawto 100,150
drawto 110,150
drawto 110,99
moveto 160,170
drawto 220,150
drawto 210,150
drawto 210,99
moveto 170,130
for j:=1 to 4 do //window
 box
 right 90
endfor j
plottext 20,20,"house & window"
plot 20,180 //star
//more stars
for i:=1 to 4 do plot 110*rnd(1),89*rnd(1)+110
for i:=1 to 4 do plot 210+109*rnd(1),89*rnd(1)+110
for i:=1 to 2 do plot 110+110*rnd(1),29*rnd(1)+170
//
proc box //window pane
 for i:=1 to 4 do
  forward 10
  right 90
 endfor i
endproc box
```

Package Maker

by Dick Klingens & Marcel Bokhorst
Dutch COMAL Users Group

One of the (many) nice things in COMAL is the possibility to link packages onto programs. However, one must have a fair knowledge of machine language to develop a package. On the other hand, most COMAL programmers are able to test a procedure in command (edit) mode.

In discussing about how to create a real COMAL compiler we started with what is now *package'maker* on *Today Disk #14*. This program, entirely written in COMAL, compiles COMAL procedures into assembler source code. The compilation is direct, so one can see in the source code how the procedure is compiled. I think this program can serve as a tool in teaching assembler programming too.

This prototype has these limitations:

- only **CLOSED** procedures can be compiled (no functions)
- only one procedure per package
- no strings or arrays are allowed as parameters or variables within the procedure
- only the following structures can be compiled:

```
WHILE .. DO
  <statements>
ENDWHILE

IF .. THEN
  <statements>
ELSE
  <statements>
ENDIF
(no one line IF or IF..ENDIF)
```

- parameters can be called by reference or value
- no **PRINT** or **IMPORT** statements
- no calling other procedures or the procedure itself
- the only built in functions that can be called are: **ATN, COS, ESC, EXP, INT, LOG, RND, SGN, SIN, SQR,** and **TAN**
- the only allowed logical operators are: <, >, <=, >=, <>, **OR, NOT,** and **AND**
- no short assignments, such as :+ and :-
- only real and integer variables are allowed
- boolean types are allowed (**TRUE** and **FALSE**)
- no binary constants (hex is possible)
- complex assignments with parentheses, functions, and the operators +, -, *, /, and ^ are allowed
- **PEEK, POKE** and **SYS** are allowed

To those interested in how the program works, you must first use the built in reveal procedure to unhide the program lines (type SCAN, then reveal). Then you can **LIST** the program. If you do this, though, you should hide the program lines before you add any procedures. Just issue the command hide(9999). (*Package'maker* skips hidden lines when searching for the merged procedure.)

When RUN, the program asks for the name of the procedure you wish to compile (the procedure must be in memory, merged into this program) what name you want the package called for the **USE** command, (it must be different than the procedure name) and the name of the output file. If you want to send the output of the compiler to the screen, enter the name **ds:**. If output is going to the screen, the listing can be paused by pressing the spacebar. If an error occurs, an error message is printed on the screen referring to the bad line. If all went well, you will see:

More ►

Compilation completed, no errors.

The program is not entirely tested, so there may be errors in it. I hope you find the program educational. Remember, this program was written as a prototype - feel free to experiment and improve it. I ask only that you share your work as we have. Since the source code generated makes extensive use of COMAL routines, you still have to have the cartridge in order to make use of any output from this program.

We used *proc.swap* with *package'maker* to get *src.test'package*. The Commodore assembler compiled it to *pkg.test'package*. All of these files are on *Today Disk #14*. The compiled COMAL procedure is not faster then the uncompiled one! Perhaps, one of the readers can tell us how this is possible. Tests with some other procedures proved once again that COMAL runs with machine speed. This is why we decided **not** to make a real COMAL compiler. The following sample procedure demonstrates the process of compiling.

[Editors note: This program is an ideal education tool for those trying to create their first COMAL package. It can also serve as a tool for those trying to learn how COMAL operates at the machine code level. Here is a challenge for anyone thinking that this program is interesting, but doesn't do anything useful. Extend the program to include the ability to compile more than one procedure, to compile functions, to handle REPEAT .. UNTIL and FOR loops, and to handle the variations in the IF structure. Who knows, this could be the first step towards a true compiler.]

Further Reference:

COMAL 2.0 Internal Structure, COMAL Today #9, page 50
COMAL 2.0 Token Table, COMAL Today #9, page 54
COMAL 2.0 Memory Map, COMAL Today #6, page 22
Packages Library by David Stidolph
COMAL 2.0 Packages by Jesse Knight

Swap real numbers

```
load "package'maker"
merge "proc.swap"
display
proc swap(ref first,ref second) closed
  third:=first
  first:=second
  second:=third
endproc swap
run
Procedure name : swap
Package name   : test'package
Source name    : lp:
;
*     = $8009
;
      .opt nolist
      .lib lib.labels
;
      .byte defpag
      .word end
      .word dummy
      .byte 12,'test''package'
      .word procs
      .word dummy
      .byte 0
;
procs .byte 4,'swap'
      .word prhead
      .byte 0
;
prhead .byte proc
      .word prcode
      .byte 2
      .byte ref+real
      .byte ref+real
      .byte endprc

prcode
;
; real variable first
;
      lda #1
      jsr fndpar
      lda copy1
      ldy copy1+1
      jsr ldac1
      jsr pusha1
;
; assign real third
;
      jsr popa1
      ldx #<loc001
      ldy #>loc001
      jsr stac1
;
; real variable second
;
      lda #2
      jsr fndpar
      lda copy1
      ldy copy1+1
      jsr ldac1
      jsr pusha1
;
; assign real first
;
      lda #1
      jsr fndpar
      jsr popa1
      ldx copy1
      ldy copy1+1
      jsr stac1
;
; real variable third
;
      lda #<loc001
      ldy #>loc001
      jsr ldac1
      jsr pusha1
;
; assign real second
;
      lda #2
      jsr fndpar
      jsr popa1
      ldx copy1
      ldy copy1+1
      jsr stac1
      rts

loc001 .byte 0,0,0,0,0
;
end   .end
```

Simple Term

by David Stidolph

This telecommunications program is short and provides only a few features: dialing, re-dialing, waiting for calls, ASCII translation, and full/ half duplex control. It takes up very little memory and can be extended. The program is listed here and is on *Today Disk #14* as well.

The program was designed from the beginning to be adapted by the user. I tried to write it in the most logical fashion I could so that you can understand it without a lot of comments. The program is under 300 lines long, so there is plenty of memory space for improvements like file transfer and bufferring.

One possible extension would be a modem configuration procedure (for parity, stop bits, etc.) that would allow changes while the program is running. As it is now, the modem is opened for 300 baud, 8 data bits, no parity, and one stop bit.

Many terminal programs I've seen had a common flaw - improper carrier detection. It seems that when you call a number and get a busy signal, the modem thinks it is another computer and switches on the carrier detect line. Terminal programs will generally assume this is a valid computer and connect the user. My program has a wait loop that makes sure it is getting a steady signal.

```
// delete "0:simple'term"
//  by David Stidolph
// save  "0:simple'term"
//
open'modem(2) // 8 bit/no parity
background 6
border 6
help'screen
//
trap esc-
dim c$ of 1, k$ of 1
dim number$ of 20
dim translation$ of 256
dim ret$ of 1, move'left$ of 1
ret$:=chr$(13)
move'left$:=chr$(157)
ascii:=true
echo:=false
connect:=false
modem'get'init(2)
build'table
hang'up
//
repeat
 pencolor 1
 modem'get
 while l<>0 do
  if ascii then
   c$:=translation$(l:l)
  else
   c$:=chr$(l)
  endif
  print'char
  modem'get
 endwhile
 l:=ord(key$+chr$(0))
 if l<>0 then
  if l>=133 and l<=140 then
   process'command
  else
   if echo then print'char
   if ascii then
    l:=(chr$(l) in translation$)
   endif
   print file 2: chr$(l),
  endif
 endif
until esc
hang'up
close
while esc do null
trap esc+
print "Done."
end
//
func carrier closed
 return not (peek(56577) mod 32) div 16
endfunc carrier
//
func ringing closed
 t:=peek(56577) mod 16
 t:=t div 8
 return not t
endfunc ringing
//
proc hang'up
 poke 56579,32
 poke 56577,0
endproc hang'up
//
proc off'hook
 poke 56579,32
 poke 56577,32
endproc off'hook
//
proc zero'time
 poke 160,0
 poke 161,0
 poke 162,0
endproc zero'time
//
func gettime closed
 ti:=peek(162)+peek(161)*256
 return peek(162)*65536+ti
endfunc gettime
//
proc pause(seconds)
 zero'time
 while seconds*60>gettime do
  null
 endwhile
endproc pause
//
proc dial(number$) closed
 z:=zone; l:=len(number$)
 zone 0
 for i:=1 to l do
  print number$(i),
  n:=number$(i) in "1234567890"
  if n then
   for pulse:=1 to n do
    poke 56577,0
    pause(.03)
    poke 56577,32
    pause(.03)
   endfor pulse
  elif number$(i)="." then
   pause(1)
```

More ►

Simple Term - continued

```
  endif
  pause(.5)
 endfor i
 zone z
 print
endproc dial
//
proc waitcarr(sec)
 zero'time
 repeat
  connect:=true
  for x:=1 to 25 do
   if not carrier then connect:=false
  endfor x
 until connect or gettime>=sec*60
endproc waitcarr
//
proc call(number$)
 hang'up
 pause(3)
 off'hook
 pause(1)
 dial(number$)
 waitcarr(15)
endproc call
//
proc modem'get
 sys 1005
 l:=peek(1023)
endproc modem'get
//
proc open'modem(f'num)
 open file f'num,chr$(6)+chr$(0),unit 2,3,read
endproc open'modem
//
proc dial'modem
 print chr$(147),chr$(14),
 input "Enter phone number: ": number$
 if number$<>"" then
  call(number$)
  if connect then
   print "Connected"
  else
   print "No connection"
   hang'up
  endif
 else
  print "Dial aborted"
 endif
endproc dial'modem
//
proc modem'get'init(f'num)
 poke 1005,162 // ldx #f'num
 poke 1006,f'num
 poke 1007,32 //  jsr chkin
 poke 1008,198
 poke 1009,255
 poke 1010,32 //  jsr getin
 poke 1011,228
 poke 1012,255
 poke 1013,141 // sta 1023
 poke 1014,255
 poke 1015,3
 poke 1016,32 //  jsr clrchn
 poke 1017,204
 poke 1018,255
 poke 1019,96 // rts
endproc modem'get'init
//
proc help'screen
 print chr$(147),chr$(14)
 print "f1)  Dial Modem"
 print "f2)  Keep Re-dialing"
 print "f3)  Connect"
 print "f4)  Toggle ASCII/Petscii"
 print "f5)  Disconnect"
 print "f6)  Wait for call"
 print "f7)  This help screen"
 print "f8)  Toggle Half/Full Duplex"
 print "STOP key to end"
 print
endproc help'screen
//
proc process'command
 case l of
 when 133
  dial'modem
 when 134
  off'hook
  print'mode("Line",1)
 when 135
  hang'up
  print'mode("Line",0)
 when 139
  wait'for'ring
 when 138
  ascii:=not ascii
  print'mode("ASCII",ascii)
 when 136
  help'screen
 when 137
  keep'dialing
 when 140
  echo:=not echo
  print'mode("Echo",echo)
 otherwise
 endcase
endproc process'command
//
proc wait'for'ring
 repeat
  print chr$(147),chr$(14),
  print "Waiting for Call"
  print "Press any key to quit"
  print
  while key$<>chr$(0) do null
  repeat
   k$:=key$
  until ringing or k$<>chr$(0)
  if k$=chr$(0) then
   print "Call Recieved, please wait..."
   off'hook
   wait'carr(15)
   if connect then
    print "Carrier detected -- Line on"
   endif
  else
   hang'up
   print "Waiting Aborted..."
  endif
 until connect or k$<>chr$(0)
 print
endproc wait'for'ring
//
proc print'mode(name$,state)
 if state then
  print name$;"mode is ON"
 else
  print name$;"mode is OFF"
 endif
endproc print'mode
//
proc keep'dialing
 print chr$(147),chr$(14),
 input "Enter Phone number: ": number$
 if number$<>"" then
  repeat
   print chr$(147),"Dialing";number$
   call(number$)
   if not connect then
    hang'up
    print "Press any key to stop"
    x:=1
    repeat
     k$:=key$
     x:+1
    until x>500 or k$<>chr$(0)
   endif
  until connect or k$<>chr$(0)
  if connect then
   print "Connected!"
  else
   print "Dialing aborted!"
  endif
 endif
endproc keep'dialing
//
proc build'table
 for x:=1 to 255 do
  translation$(x):=chr$(x)
 endfor x
 translation$(8):=chr$(20)
 translation$(12):=chr$(147)
 for x:=65 to 90 do
  translation$(x):=chr$(x+128)
  translation$(x+32):=chr$(x)
 endfor x
endproc build'table
//
proc print'char
 if c$<>ret$ then
  print c$,chr$(175),
  poke 212,0
  print move'left$,
 else
  print " "
  print chr$(175),move'left$,
 endif
endproc print'char
```

Terminal

article by Richard Bain
program by Robert Shingledecker

Several *COMAL Today* readers have asked about using modems with COMAL. Long before I started working here, I fell in love love with a COMAL 0.14 terminal program by Robert Shingledecker. I hope my experiences will prove useful to some of you, and that Mr. Shingledecker will accept this as a late, though deeply felt, thank you.

I was taking computer science classes at the University of Maryland. The computer terminals on campus were constantly busy, and anyone with a home terminal had a real advantage. I had a computer and a modem, but I had trouble getting a good terminal program. I tried using a few commercial programs, but didn't like waiting for long load times and being forced to sift through several menus. Besides, I couldn't afford to replace my disk drive every time the copy protection schemes went crazy. Then I went to my local users group for public domain programs. They worked well enough, but it seems someone forgot to inform the university's sophisticated mainframe computers about the Punter protocol. Then I came across the COMAL program.

Terminal is not the program to end all programs. It's just a dumb terminal program; it can send and receive characters and perform the ASCII translations, but that is all. But what exactly did I want?

The COMAL program had two things going for it. It let me use my computer as a terminal to the school's computer without making any changes to the program. Since I could do this from home without fighting the crowds on campus, I had reason enough to use the program. The other advantage the program had was that it was written in COMAL with very minor machine language subroutines. This made it easy to change and adapt to my needs.

The program was short, well structured, and easy to understand. It wasn't cluttered with dialing routines or time consuming menus. After a little trial and error, I was able to merge in a routine to upload files. This wasn't easy, because the disk drive and the modem don't get along very well. To make matters worse, using the disk drive caused the some machine language subroutines in the cassette buffer to be overwritten. When I solved these problems, I discovered I was sending characters faster than the other computer could receive them. A short delay fixed that.

I could use my word processor to write my assignments on, and then upload them to the school's computer saving countless *on line* typing time and errors. Adding downloads to the program had to wait until I got the COMAL cartridge. COMAL 0.14 didn't have enough memory for a buffer, and even though I could handle disk access during uploads, I couldn't do it for downloads.

Why do I like Robert Shingledecker's program so much? Not because it is the program to **end** all programs, but because it is the program to **begin** all programs. Even if a program does exactly what you want it to do, it may not quite do what someone else wants it to do. I like COMAL because it allows good programs to easily be customized to a variety of needs.

Terminal is on *Today Disk #14*. Make sure to set **id$**, **password$**, and **off$** to proper values for your use.

More ►

```
// save  "0:terminal
// by Robert Shingledecker
print "Please wait - setting up..."
poke 659,6 // rs232 control
poke 660,16 // rs232 command
open file 5,"",unit 2,0,read
get'rs232'init
dim a$ of 1, id$ of 10, password$ of 10
dim off$ of 3, f#(0:255), t#(0:255)
id$:="77770,101"
password$:="free-demo"
off$:="off"
set'up'tables
print chr$(147),chr$(17),
print "Terminal Program Functions:"
print "  f1 = CNTRL C - Call remote."
print "  f3 = Transmit User ID."
print "  f5 = Transmit Password."
print "  f7 = LOGOFF remote system."
print "Press STOP key to end program."
trap esc-
repeat
 x:=get'rs232(5)
 if x<>0 then
  print " ",chr$(157),chr$(f#(x)),
  if f#(x)=34 then poke 212,0
 else
  print chr$(rv)," ",chr$(157),chr$(146),
  a$:=key$
  x:=ord(a$+chr$(0))
  if x=134 then
   print file 5: id$
  elif x=135 then
   print file 5: password$
  elif x=136 then
   print file 5: off$
  elif x=0 then
   null
  else
   print file 5: chr$(t#(x)),
  endif
 endif
 ct:+1
 if ct=6 then
  ct:=0
  rv:=164-rv
 endif
until esc
close
end
//
func get'rs232(file'num) closed
 poke 830,file'num
 sys 829
 return peek(828)
endfunc get'rs232
//
proc get'rs232'init closed
 for memory:=828 to 843 do
  read value
  poke memory,value
 endfor memory
 data 0,162,0,32,198,255,32,228
 data 255,141,60,3,32,204,255,96
endproc get'rs232'init
//
proc set'up'tables
 for j:=32 to 64 do t#(j):=j
 t#(13):=13; t#(20):=8; rv:=18; ct:=0
 for j:=65 to 90 do t#(j):=j+32
 for j:=91 to 95 do t#(j):=j
 for j:=193 to 218 do t#(j):=j-128
 t#(133):=3
 for j:=0 to 255 do
  k:=t#(j)
  if k<>0 then
   f#(k):=j
   f#(k+128):=j
  endif
 endfor j
endproc set'up'tables
```

UNITED STATES COMMODORE COUNCIL

P.O. BOX 2310. ROSEBURG. OR 97470 (503) 673-2259

JOIN AMERICA'S LARGEST COMMODORE USERS SUPPORT GROUP

INITIAL DUES $25.00/RENEWAL $20.00

BENEFITS

* ACCESS TO THOUSANDS OF PUBLIC DOMAIN PROGRAMS.
* ONE YEAR SUBSCRIPTION TO USCC COMMAND PERFORMANCE.
* CONSUMER ASSISTANCE.
* TECHNICAL ASSISTANCE.
* FREE UTILITY PROGRAMS
* VIC-20 C64 C128 AMIGA

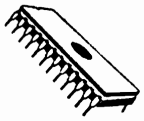
USCC COMMAND PERFORMANCE

PUBLISHED BI-MONTHLY
PRODUCT EVALUATIONS
SOFTWARE REVIEWS
HOW TO FEATURES
PROGRAMMING INFORMATION
LATEST PRODUCT INFORMATION
INFORMATIVE FACTS

BONUS ** FREE VIEWTRON STARTER KIT & 1 HOUR ON-LINE TIME

More Modem Fun

by Richard Bain



Super Chip has two new commands to make writing terminal programs a snap: countsp and getsp$. The short terminal program at the end of this article uses getsp$.

Getsp$ is a string function similiar to the COMAL command get$. It has two parameters. The first is a file number. It must be the number of a file that has already been opened to the modem port. The second parameter must be a number between 0 and 256 inclusive. Different numbers have special meanings:

If the second parameter is 256, getsp$ will return all the characters in the RS232 buffer without waiting for any more. If there aren't any, a null string will be returned. If you assign getsp$ to another string, make sure that string is dimensioned to 255, insuring that no characters are lost.

If the second parameter is 0, getsp$ will return **one** character unless the buffer is empty. (It then returns a null string).

If the second parameter is between 1 and 255 inclusive, getsp$ will return that number of characters. Be careful using this setting. Getsp$ will wait forever for the characters, if none are being received, just like the get$ command. Getsp$(filenumber,256) was written specifically to avoid this delay.

Countsp is a function which returns the number of characters in the RS232 buffer. It is not needed with getsp$, but some programmers may still want to use it.

```
if countsp > 200 then print "RS232 buffer almost full"
```

The RS232 buffer is limited to 255 characters and it may overflow. If this happens, data will be truncated, but no error results. Read from the RS232 buffer as often as possible to avoid losing characters.

Opening a file to the modem port is **not** enough to make it receive data. You must also *tell it to start listening.* Use getsp$(filenumber,256) to do this. The first time you issue this command, you will get the null string because the modem has not been listening. Countsp will return 0 until you issue getsp$, even if the modem is receiving data. After issuing getsp$ once, the modem port starts listening for data; issue getsp$ again to retrieve it.

Two computers can run the following program to chat with each other. It even uses different text colors for each computer's characters (white is for data received by modem - your characters typed are in black). The program might also work with another computer using a different terminal program.

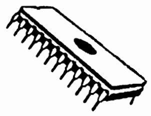
```
USE colors
USE files
USE graphics
textbackground(grey)
OPEN FILE 2,"sp:d8b300/a-/l-"
DIM buffer$ OF 255
LOOP
  textcolor(white) //incoming characters
  buffer$:=getsp$(2,256) // other computer
  IF buffer$>"" THEN PRINT buffer$,
  buffer$:=KEY$ // your keyboard
  IF buffer$>"" THEN
    textcolor(black) //your characters typed
    PRINT buffer$,
    PRINT FILE 2: buffer$,
  ENDIF
ENDLOOP
```

Further Reference:

2.0 Modem Update, COMAL Today #11, page 38
Modem Fun With COMAL 2.0, COMAL Today #9, page 10 ▣

Input For Modems

by David Stidolph



One of the nicer features of the COMAL 2.0 cartridge is the addition of protected INPUT fields. This feature allows the programmer to set up an area that the user can type in, but cannot cursor out of. I wanted the same feature for a modem program I am working on, and developed this string function. It is fairly complicated, and requires certain variables be initialized outside of the function.

PARAMETERS

In order to call get'input$ you need to pass it 5 parameters.

default$ is any text you want the user to have "already typed in." This could be the name *new* when asking for a **user id** so that new users would simply press *<return>*.

max'len is the length of the input field in characters. Once the field has been filled, only the delete character and the carriage return are recognized.

valid$ is a string containing all the characters you want the users to be able to type in. This could be just numbers, lower case, 'Y' and 'N' for Yes/No questions, etc.

modem is the file number the modem was opened with.

max'time is the length of time, in seconds, you wish to allow the user to type before a trapable error (#300, "out of time") is generated. This allows you to prevent a single user from simply walking away from his computer and tying yours up all day.

OUTSIDE VARIABLES

To make the function more useful to all systems, the string variables **del$**, **cr$**, and **ff$** are imported.

**del$** is the delete character. On Commodore systems this is CHR$(20).
**cr$** is a carriage return. On almost all computer systems it is CHR$(13).
**ff$** is a form feed character. On Commodore systems this is CHR$(147).

```
FUNC get'input$(default$,max'len,valid$,
modem,max'time) CLOSED // wrap line
  IMPORT del$,cr$,ff$
  USE files
  DIM c$ OF 1, string$ OF max'len
  string$:=default$
  PRINT FILE modem: string$,
  TIME 0
  REPEAT
    REPEAT
      c$:=getsp$(modem,0)
      IF TIME>max'time*60 THEN
        REPORT 300,"Out of time"
      ENDIF
    UNTIL c$>""0""
    CASE c$ OF
    WHEN del$ // delete character
      str'len:=LEN(string$)
      IF str'len>1 THEN
        string$:=string$(:str'len-1)
        PRINT FILE modem: del$,
      ELIF str'len=1 THEN
        string$:=""
        PRINT FILE modem: del$,
      ENDIF
    WHEN ff$ // clear screen
      FOR x:=1 TO LEN(string$) DO
        PRINT FILE modem: del$,
      ENDFOR x
      string$:=""
    WHEN cr$ // carriage return
      NULL
    OTHERWISE
      IF LEN(string$)<max'len AND c$ IN valid$ THEN
        string$:+c$
        PRINT FILE modem: c$,
      ENDIF
    ENDCASE
  UNTIL c$=cr$
  RETURN string$
ENDFUNC get'input$
```

Super Chip Update

A few of our readers thought that Super Chip was for the C128 only. However, Super Chip adds over 80 commands to COMAL that work equally well on the C64 and the C128. Only the 29 added C128 package commands are restricted to use on the C128.

There are too many commands in Super Chip to go into detail about all of them. Perhaps someone could submit an article on using it. Or maybe a small book is needed. Your letters and programs will be especially welcome in this regard. This issue highlights **drive9** and **type** from the **files** package, and **lock'lowercase**, **lock'uppercase** and **unlock'case** from the **keyboard** package. Other notes are included after these highlights.

Drive9 will be one one of the first commands users with two 1541 disk drives (or an SX64 with a second 1541) will use. The **drive9** command will allow the use of the second disk drive as unit 9. It then is accessed by placing "2:" before the name of the file you are using. Examples:

```
CAT "2:"         //view catalog
LOAD "2:NAME"    //load program
```

To use the **drive9** command for a second disk drive, follow these steps:

1) Turn on the computer and the disk drive you want as unit 9. Leave the other disk drive turned off.
2) Type: **USE files**
   drive9
3) Now turn on your unit 8 disk drive.

Drive9 will work with 1541 and MSD disk drives. If you have a 1571, use the hardware switch instead. Once the drive number is changed, it will remain changed until the disk drive is turned off or another command changes it again. Going to BASIC or using a reset button on the computer will not change the drive number back to 8. If you want to change unit 9 back to unit 8, use the **drive8** command. One final note: the PASS command will work with your unit 9 drive. Type the command as usual, but add a ",9" at the end:

```
PASS "n0:workdisk,wd",9
```

The **type** command may prove to be the most popular command in Super Chip. In its simplest form, it is used to type the contents of a sequential file onto the text screen. If the printing is too fast, use the space bar to pause it. If you don't want to see the entire file, hit the stop key. Output is not limited to the text screen. It will go to whatever file has been selected with the **SELECT OUTPUT** command, even the 80 column screen ("u7:") on the C128. We have used the type command successfully to print to our laser printer via the RS232 port at 2400 baud. After typing a file, be sure to issue the command **SELECT "ds:"** to direct output back to the normal screen and close the **SELECT OUTPUT** file.

The file being typed does not have to be a sequential file, but to use a program file, you must include a ",p,r" at the end of the filename. Files that type well include *Easy Script* files, *PaperClip* files (control z), and listed COMAL programs. *Speedscript* files can't be typed to the screen, because the **type** command does not convert screen poke codes or control characters. Here is an example to copy a program file from unit 8 to unit 9:

```
USE files // unit to unit copier
SELECT "2:filename,p,w" //unit 9
type ("0:filename2,p,r") //unit 8
SELECT "ds:" // all done
```

More ►

Finally, the keyboard package lets you lock in either uppercase or lowercase mode at any time. Many programs have formatted output which requires the upper/ lower case mode or the upper case/ graphics mode. The screen can be ruined if the user presses the *<shift>* and *<commodore>* keys together. To protect against this, use the **lock'uppercase** or **lock'lowercase** command. Super Chip starts with the lower case mode locked in. To enable the *<shift+commodore>* keys use the **unlock'case** command.

SUPER CHIP NOTES

On page 72 of *COMAL Today #13* the examples for **quicksort** only say *sort*. The correct command is **quicksort**:

quicksort(<array$()>,<integer>,<integer>)
quicksort(name$(),1,last)

The example for the **colors** package on page 72 of *COMAL Today #13* was correct, but it assumed the graphics package was in use. A **USE graphics** or **USE turtle** command must be issued prior to trying the example. That example only alters the graphics screen, not the text screen.

James Synnamon was the first person to report that the C128 **no scroll** key stopped a program listing (or disk catalog) from scrolling. Most of the other keys on the top row also work. Press the **help** key while editing a line, and it is returned to its original unindented form (just like *control a*). The **alt** key acts like the insert key and **linefeed** acts like cursor down. The **caps lock** and **40/80 display** keys are not recognized by Super Chip. From program mode, the keys return these codes:

Key	ORD	Key	ORD	Key	ORD
alt =	148	help =	1	linefeed =	17
esc =	3	tab =	162	no scroll =	32

The **c128** package uses the cassette buffer. If this buffer is needed for other uses, use this new command to free the buffer:

discard'c128

We have been told that Super Chip is compatible with the *Buscard*, but not with the *Handic IEEE interface*. More comments are welcome.

COMAL Today #13 page 70 states that the range of the **even** command in the **math** package is *-32768 to 32767*. **Even** and **odd** have both been updated to work for all real numbers.

COMAL Today #13 page 63, mentions that the Auto Start system has a short delay added to let you insert a disk after power is turned on. The added delay has been removed at the request of our testers, but there is still enough of a delay during computer and chip initialization to insert a disk after power up.

Super Chip includes an updated **rabbit** package. **Rabbit** is a fastloader for the 1541 disk drive only. The commands are the same as documented in *COMAL Today #12*, but Phil Bacon modified it (with *sizzle* code). You now may have a printer connected and the screen no longer blinks. **Rabbit** is not enabled on power up, but the commands to do so are printed on the startup screen. Cursor up to them and hit *<return>* twice.

Rabbit uses the RS232 buffer. If you need to use the RS232 port for any reason, first disable **rabbit** with the **setfast(false)** command. If you use the **bread** and **bwrite** commands, you **must** use buffer 0. **Device** and **setdevice** refer only to which device will be in fast mode. They are not related to the **drive8** and **drive9** commands in the **files** package.

More ►

Please note a peculiarity of the fast loader in Super Chip. You can load a program from direct mode, but do **not** have the graphics screen displayed when you issue the **LOAD** comand. Chaining a program will return you to the text mode:

```
1 CHAIN "name"
RUN
```

Also, please note a special feature of Auto Start. When the autostarted program ends, it is automatically erased from memory. This feature allows you to begin programming with a *clean slate*. But, beware of a side effect of this. If the autostarted program chains another, the final program to **END** or **STOP** is erased. To see how this can have an unexpected result, try autostarting the *hi* program on *Today Disk #13*. Choose *chip.testchip* from the menu. Then reply "y" to the demo request. Now, you are in the third program. Choose to demo **system2**. When it attempts to show you how **reveal** works, the program is erased and the demo is lost. The moral of the story is: don't autostart programs which need to stay in memory after ending.

It's been over a month since we began distributing Super Chip. A few people returned their chip & cartridge to avoid problems installing it. Actually, we highly recommend that you allow us to install the chip for you. We then test it to make sure it is funtioning properly in your cartridge. Since you then must be without your cartridge for some time, we try to install the chip, test it, and ship it back to you within two days. This should avoid most withdrawl symptoms. Of course, if you order a cartridge at the same time as a Super Chip, we automatically install and test it for you.

At this time, it looks like the Super Chip is a success. Yes, it is a bit skimpy on documentation, but articles and notes in *COMAL Today* will help. And perhaps in a few months a small Chip book can be written. Would you be interested in it?

Thus far, only one "quirk" has been reported in Super Chip. Robert Bain found that an **input80$** on the bottom line of the 80 column screen causes the screen to scroll. If scrolling is not desired, avoid **input80$** on the bottom line.

It's not the chips fault, but when we ran the *chip.1000primes* program on *Today Disk #13* using the C128 80 column screen, we noticed something strange. Every other line of numbers ended in the middle. This caused a mild panic and David Stidolph searched his C128 package code for the problem. However, Robert Ross quickly found the reason in the COMAL ROMs. It seems that COMAL prints a carriage return on any line that has more than 132 characters (except on the 40 column screen). It actually prints the carriage return at the last possible break so that it doesn't split a word or number in half. This is useful for anyone with a 132 column printer, but is annoying in all other cases. Any suggestions?

One final observation. Commodore power supplies have been a constant source of problems. Finally, it seems that they are a sore spot even for Commodore themselves. We hear that now when you purchase the 512K memory expansion module, a new heavy duty power supply is included at no extra charge. If the power supply exists, we hope it can be purchased separately. If anyone has further information on this, please let us know right away. Our C128 needs a new heavy duty power supply! ■

Epson Package

by Green, Rose, Wright, and Grainger
U.K. COMAL USER GROUP

PKG.PLLPRT is a COMAL 2.0 package (on *Today Disk #14*) for the Epson, or look-a-like, printer fitted with a Centronics parallel interface connected to the user port.

Use on the Epson FX80 and Kaga Printer

STEP 1: issue the following commands:

```
LINK "pkg.pllprt"
USE pllprt
USE graphics
USE system
setprinter("u4:/a+")
```

The last two lines provide ASCII conversion so the printer will print text normally (but not graphics).

STEP 2: Dump the graphics screen using one of the two package commands below:

hr0prt dumps the hi-res screen.
hr1prt dumps the multicolor screen.

The screen is printed sideways with a 400 X 640 dot array using the 576 X 8 dot mode on the printer. The grey scales for each color of the same density have a different bit pattern. Thus adjacent patches of color look different.

Use on the Epson RX80

The Epson RX80 supports a 640 by 8 dot mode. This means that hi-res images will be distorted on this printer. One solution is to modify the hi-res screen x-coordinate system by 319*640/576 (0-354) before you draw it (use the WINDOW command). The result will look peculiar on the screen but gives a good printout.

You also must alter the package to select the correct graphic mode on this printer. Change STEP 1 to:

```
LINK "pkg.pllprt"
POKE $81f3,4
USE pllprt
USE graphics
USE system
setprinter("u4:/a+")
window(0,354,0,199) // settings vary
```

Use on the Epson MX80

Use the following procedure for STEP 1 to set the MX80 in single density graphics mode. Note the window command is different.

```
LINK "pkg.pllprt"
POKE $81f2,75
POKE $81f3,184
POKE $81f4,1
POKE $8126,4
POKE $812a,40
USE pllprt
USE graphics
USE system
setprinter("u4:/a+")
window(0,266,0,199) //setting varies
```

You may achieve a better picture by using double density graphics mode using this for STEP 1:

```
LINK "pkg.pllprt"
POKE $81f2,76
POKE $81f3,232
POKE $81f4,1
POKE $8126,4
USE pllprt
USE graphics
USE system
setprinter("u4:/a+")
window(0,532,0,199) //setting varies
```

Wumpus

by Marc Clifford

Here is an old computer game. It is a predecessor of modern adventure games. A version for both COMAL 0.14 and 2.0 is on *Today Disk #14*. The 2.0 version is listed below.

```
PAGE
PRINT AT 5,11: "Welcome to Wumpus!"13""13""
intro
IF askinstruct THEN doinstruct
initialize
REPEAT
  doaturn
UNTIL gameover
END "Bye"
//
PROC doaturn
  describe
  PRINT
  REPEAT
    INPUT AT 0,1,1: "(m,s,q,?)> ": c$,
  UNTIL c$ IN command$ AND c$>""
  PRINT
  CASE c$ OF
  WHEN "m","M"
    domove
  WHEN "s","S"
    doshoot
  WHEN "q","Q"
    doquit
  WHEN "?"
    doinstruct
  ENDCASE
ENDPROC doaturn
//
PROC initialize
  maxrooms:=20; arrow#:=7
  DIM bat$ OF 2, pit$ OF 2
  DIM command$ OF 7, c$ OF 1
  DIM room$(maxrooms) OF 10
  DIM shots(maxrooms)
  command$:="mMsSqQ?"
  FOR x:=1 TO maxrooms DO room$(x):=""
  FOR thisroom:=2 TO maxrooms DO
    add'tunnel(thisroom,thisroom-1)
  ENDFOR thisroom
  FOR thisroom:=3 TO maxrooms DO
    newtunnel:=RND(1,thisroom-2)
    IF NOT included(newtunnel,room$(thisroom)) THEN
      add'tunnel(thisroom,newtunnel)
    ENDIF
  ENDFOR thisroom
  wumpus#:=RND(1,maxrooms)
  bat$:=CHR$(RND(1,maxrooms))+CHR$(RND(1,maxrooms))
  pit$:=CHR$(RND(1,maxrooms))+CHR$(RND(1,maxrooms))
  REPEAT
    player#:=RND(1,20); ok:=TRUE
    IF player#=wumpus# THEN ok:=FALSE
    IF included(player#,bat$) THEN ok:=FALSE
    IF included(player#,pit$) THEN ok:=FALSE
  UNTIL ok
  quitting:=FALSE; killed:=FALSE; wumpdead:=FALSE
  //
  PROC add'tunnel(r1,r2)
    room$(r1):+CHR$(r2)
    room$(r2):+CHR$(r1)
  ENDPROC add'tunnel
ENDPROC initialize
//
FUNC askinstruct CLOSED
  DIM r$ OF 1
  INPUT "Do you want instructions (y/n): ": r$
  RETURN r$ IN "yY"
ENDFUNC askinstruct
//
FUNC gameover
  done:=quitting OR killed OR wumpdead OR (arrow#=0)
  RETURN done
ENDFUNC gameover
//
FUNC included(i,set$) CLOSED
  DIM test$ OF 1
  test$:=CHR$(i)
  RETURN test$ IN set$
ENDFUNC included
//
FUNC intersect(a$,b$) CLOSED
  it:=FALSE
  FOR x:=1 TO LEN(a$) DO
    IF a$(x) IN b$ THEN it:=TRUE
  ENDFOR x
  RETURN it
ENDFUNC intersect
//
PROC describe
  PAGE
  PRINT AT 10,10: "You are in room ",player#
  PRINT AT 12,5: "There are tunnels to rooms:"
  PRINT
  FOR i:=1 TO maxrooms DO
    IF included(i,room$(player#)) THEN PRINT i;"";
  ENDFOR i
  wumpnear:=FALSE
  PRINT
  PRINT
  IF included(player#,room$(wumpus#)) THEN
  wumpnear:=TRUE // wrap line
  IF intersect(room$(player#),room$(wumpus#))
  THEN wumpnear:=TRUE // wrap line
  IF wumpnear THEN PRINT "I smell a wumpus"
  IF intersect(bat$,room$(player#)) THEN
  PRINT "I hear bats" // wrap line
  IF intersect(pit$,room$(player#)) THEN
  PRINT "I feel a draft" // wrap line
  PRINT
ENDPROC describe
//
```

More ►

```
PROC doquit
  DIM a$ OF 1
  PAGE
  INPUT AT 10,1,1: "Do you really want to quit now? ": a$
  quitting:=(a$ IN "yY")
ENDPROC doquit
//
PROC domove
  PRINT
  INPUT "Move to room#: ": dest
  IF dest<1 OR dest>maxrooms THEN
    PRINT "There is no room #",dest
  ELIF NOT included(dest,room$(player#)) THEN
    PRINT "I see no tunnel to room #",dest
  ELSE
    player#:=dest
  ENDIF
  checkbats; checkwump; checkpits
ENDPROC domove
//
PROC checkbats
  IF included(player#,bat$) THEN
    REPEAT
      flewto:=RND(1,maxrooms)
    UNTIL NOT included(flewto,bat$+pit$) AND
    flewto<>wumpus# // wrap line
    player#:=flewto
    PRINT "A superbat picked you up and carried you off."
    FOR delay:=1 TO 2000 DO NULL
  ENDIF
ENDPROC checkbats
//
PROC checkwump
  newwump:=RND(1,maxrooms)
  IF included(newwump,room$(wumpus#)) THEN
  wumpus#:=newwump // wrap line
  IF wumpus#=player# THEN
    PRINT "Look out!! The Wumpus got you!"
    PRINT ""13"Better luck next time."
    killed:=TRUE
  ENDIF
ENDPROC checkwump
//
PROC checkpits
  IF NOT killed AND included(player#,pit$) THEN
    PRINT "Don't do that! Too late, you fell"
    PRINT "into a pit. You should be more careful"
    killed:=TRUE
  ENDIF
ENDPROC checkpits
//
PROC doshoot
  PRINT
  PRINT "How many rooms do you wish to shoot"
  INPUT "your arrow through: ": numr
  FOR x:=1 TO numr DO
    PRINT x,") room #",
    INPUT "": shots(x)
  ENDFOR x
  lastroom#:=player#; x:=1
  WHILE x<=numr DO
    shotat:=shots(x)
    IF NOT included(shotat,room$(lastroom#)) THEN
      shotat:=randroom(room$(lastroom#))
    ENDIF
    lastroom#:=shotat
    IF wumpus#=shotat THEN wumpdead:=TRUE
    IF player#=shotat THEN killed:=TRUE
    IF wumpdead OR killed THEN x:=numr
    x:+1
  ENDWHILE
  arrow#:-1
  IF killed THEN
    PRINT "You Klutz! You just shot yourself."
  ELIF wumpdead THEN
    PRINT "Congratulations!"
    PRINT "You slew the fearsome Wumpus!"
  ELIF NOT arrow# THEN
    PRINT "You ran out of arrows."
  ELSE
    PRINT "You missed!"
    FOR delay:=1 TO 2000 DO NULL
  ENDIF
ENDPROC doshoot
//
FUNC randroom(limit$)
  REPEAT
    possible:=RND(1,maxrooms)
  UNTIL included(possible,limit$)
  RETURN possible
ENDFUNC randroom
//
PROC doinstruct
  PAGE
  PRINT "You will be told what room you"
  PRINT "are in and the rooms connected to it "
  PRINT "by tunnels. Bats can be heard if they"
  PRINT "are in adjacent rooms. Pits in "
  PRINT "adjacent rooms cause a draft."
  PRINT
  PRINT "The Wumpus can be smelled if he is"
  PRINT "closer than 4 connected rooms away."
  PRINT
  PRINT "When you shoot, you can fire your"
  PRINT "arrow through several connected rooms"
  PRINT "by choosing the number of rooms you "
  PRINT "wish to shoot through, and then giving"
  PRINT "the rooms in the order in which the "
  PRINT "arrow will reach them. If you make a"
  PRINT "mistake, the arrow may go anywhere."
  PRINT
  PRINT TAB(12),"Press RETURN"
  WHILE KEY$<>""13"" DO NULL
ENDPROC doinstruct
//
PROC intro
  PRINT "Your mission is to hunt for the Wumpus"
  PRINT "in his cave. To succeed, you must shoot"
  PRINT "it with one of your 7 arrows."13""
  PRINT "There are bats in the cave that may"
  PRINT "pick you up and place you in another"
  PRINT "room. If you enter a room which has a"
  PRINT "pit, you will fall into it."13""
  PRINT "If the Wumpus finds you, or you run out"
  PRINT "of arrows, you lose."13""
ENDPROC intro
```

Two Guessing Games

GUESS MY WORD

by Lew Fleishman

When I received my copy of the *Cartridge Tutorial Binder* I became fascinated by Program 10, a word guessing game. As I continued to modify this program it evolved into *Guess My Word*. A few explanations are warranted.

This game is for 2 - 5 players. Point selection is limited to 21 to 150 points.

Estimated level of play lets the program select the number of words needed to reach the number of points selected. If the level of play is inappropriate, you can adjust it by increasing **factor** for fewer words, or decreasing **factor** for more.

As each player enters a word list, they may change or correct a word by typing y to the prompt: *Any Changes (y/n).*

Once the game begins, the procedure **select'list** assures that a player will not receive a word from their own word list. Once a word has been selected, it is eliminated from the word list and can't be used again. As each letter is selected, it is eliminated from the alphabet listing shown below the letter selection area. If a player tries to guess a letter that has already been selected, there is no penalty. If a word contains several letters that are the same, all identical letters are shown at once. When this happens it is possible to receive a score of more than 20 (a perfect score) for a given word. Minus points are possible; random guessing is discouraged.

As each word is properly guessed, the score received for that word is shown. At the end of each round all players' scores are displayed. Procedure **game'over** will detect if enough points have been scored to end the game, and then show all scores and indicate the winner if necessary.

If the game exhausts the original word list, the procedure **enter'more'words** will allow additional words to be added. If more than two words are needed to complete the game, the procedure will recommend a lower level of play for the next game.

Additional games may use the same set-up (players, points, level) or a new set-up can be requested.

Further Reference:

Cartridge Tutorial Binder, page 66 ■

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

GUESS MY CODE

by Tony Granata

Guessmycode is a short game on *Today Disk #14* based on a game I played in grade school (I don't remember the name). It is a challenging and fun game for adults as well as children. The object of the game is to crack a code consisting of four colored stars. Each star can be one of six colors; no color will be duplicated in the code. On each turn, the somewhat conceited computer will ask for your guess. It will evaluate your guess by telling you how many of your colors are in the code, and how many are in the correct position in the sequence. When the code is cracked, the computer will evaluate your performance. *Guessmycode* was written in COMAL 2.0 and converted into COMAL 0.14. I hope you enjoy the game. *[Editors note: the stars have been changed to letters to enhance play on monochrome monitors.]* ■

Extra Disk Programs

OLD MANSION

Marc Clifford gave us a wonderful new COMAL 2.0 adventure game. *Old'Mansion*, on *Today Disk #14*, leaves you lost in an old mansion. There are many treasures to find, but you also must get out - and the door out has been blocked shut. Can you get out alive? The program uses the graphics screen to draaw each room as you enter it. It is obvious that there is an upstairs, but how do you get there? And at first glance you might not realize that there are more than two floors in this mansion. You will need the various items you find in some of the rooms, so take them all with you. This program also features a "pop over". Just hit the STOP key to get a help screen.

Further References

Kastle, Today Disk #13, 2.0 side
Pop Over Menus, COMAL Today #11, page 18

BRIDGE HAND EVALUATOR

Today Disk #14 includes a COMAL 0.14 program by David Lee Powell which shuffles a deck of 52 cards, deals them out into four hands, and then evaluates them as bridge hands. It can also accept a hand from the keyboard to evaluate.

Hand evaluation is by the method given in the article *4 C's* in the October, 1982, issue of *The Bridge World*. (The article explains that the title's *4 C's* means: Caution! Complex Computer Count).

When entering hands, the program solicits each suit in order, expecting a card symbol **(a, k, q, j, 1 for 10, 2-9)** or *<return>* to go on to the next suit. The program ignores other keys, but does not check for 13-card hands.

The display:

The first column shows the number of cards in each suit. The next cloumn shows the honor count (basically A=3, K=2, Q=1, but with complications). The third shows the suit quality count (suit length times high card count over ten, plus complications). The fourth column shows the distribution count (void=3, singleton=2, doubleton=1, but 4-3-3-3 distribution counts minus one-half. The first point is discounted in any other distribution). The columns are totaled on the bottom line for each hand.

WINDOW MAGIC

Today Disk #14 has a machine language routine for COMAL 0.14 to scroll any size window anywhere on the text screen in any direction. David Warman included two versions: one that loads in right before the error messages in the $C000 block, and another that loads into the RS-232 buffer. If you use the first version, you are restricted to using sprites 0 through 6. The demo program *demo/window* uses this version. He is currently working on converting the window routines into a package for 2.0.

PLANET EARTH

This COMAL 0.14 program draws an outline of the United States on a globe. It will work on either the multi-color or hi-res graphics screens. The map is nice, and even includes some of the great lakes.

FANCY LETTERING

This COMAL 0.14 program demonstates how to put some nice looking large letters on the graphics screen.

C CURVE

This is an interesting pattern derived from a formula creating a C. Similar to the *Hilbert* curve, the *C Curve* has increasingly complex levels, each based on a previous level.

SQUARE CLOCK

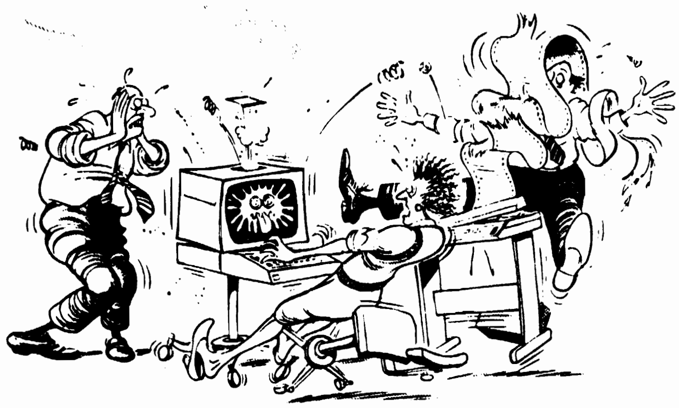
This is an interesting new clock program. It is different from the others we have released. It provides a nice way to turn your C64 into a clock.

UNIFORM POLY

Sid Seiferlein of Muskegon, MI provided us with this program to draw polygons on the graphics screen.

MORE STUDENT PROGRAMS

More student programs are on the disk than those listed on pages 27-31. The *house* program has an animated window! Also find a rocket, graph, and animated *HI.*



If you do find mistakes in this newsletter, please consider that they are there for a purpose. We try to write something for everyone, and some people are always looking for mistakes! - reprinted from DeRidder Computer Club Small Byte Chronical, June 1986

Puzzle

by Captain COMAL

Way back in grade school, I had a very nice plastic puzzle. It was a square divided into 16 smaller numbered squares, arranged 4 by 4 --- with the 16th small square missing. The object was to slide the small square pieces around for a while, so that they were in random locations. Then try to put the puzzle back into numeric order.

Last weekend, it occurred to me that I had never seen that puzzle on a computer. Perfect project! So I sat down and created a quick working version, using a joystick to *move* the pieces. The program is listed at the end of this article.

As with most programming projects, I was not satisfied with the puzzle in it's original *simple* form. I then modified it twice. It ended up on the graphics screen.

I had originally hoped to change the puzzle one more time, so that it used sprites for the 15 squares. My plan was to take a nice hi-res picture, break it into 16 squares, and use those squares for the puzzle. To do this, I needed *sprite'maker* (which is discussed in the next article).

But then I had an even better idea. Why not just provide the puzzle as it is, plus the tools to create sprites from pictures? Then challenge *COMAL Today* readers to make the next generation *puzzle.*

So, the challenge is out. Can you do it? Send your entries to us as soon as you can. We will use the best entry in a future *COMAL Today.* I am listing my original *puzzle* here. The third version of it is on *Today Disk #14.* It includes about a dozen line changes or additions.

More ►

```
    init
    solve
    PRINT AT 10,10: "WINNER"
    END ""
    //
==> PROC init
!      USE joysticks
!      dims
!      move'count:=0 // for final goal
!      display'goal
!      shuffle
!      move'count:=0 // reset to 0
!      display'board
--> ENDPROC init
    //
==> PROC solve
!   ==> REPEAT
!   !      get'move
!   !      move
!   !      display'board
!   --> UNTIL winner
--> ENDPROC solve
    //
==> PROC display'board
!      PAGE
!   ==> FOR rows:=1 TO 4 DO
!   !      PRINT USING "## ## ## ##": p(rows,1),
!   !      p(rows,2),p(rows,3),p(rows,4) //wrap line
!   --> ENDFOR rows
!      PRINT AT 23,28: "Moves:";move'count
--> ENDPROC display'board
    //
==> PROC dims
!      DIM p(1:4,1:4), stick(0:8)
--> ENDPROC dims
    //
==> FUNC winner
!      counter:=0
!   ==> FOR rows:=1 TO 4 DO
!   !   ==> FOR cols:=1 TO 4 DO
!   !   !      counter:=(counter+1) MOD 16
!   !   !      IF p(rows,cols)<>counter THEN RETURN FALSE
!   !   --> ENDFOR cols
!   --> ENDFOR rows
!      RETURN TRUE
--> ENDFUNC winner
    //
==> PROC display'goal
!      counter:=0
!   ==> FOR rows:=1 TO 4 DO
!   !   ==> FOR cols:=1 TO 4 DO
!   !   !      counter:=(counter+1) MOD 16
!   !   !      p(rows,cols):=counter
!   !   --> ENDFOR cols
!   --> ENDFOR rows
!      display'board
!      PRINT AT 22,9: "This is the final goal"
--> ENDPROC display'goal
    //
==> PROC shuffle
!      PRINT AT 24,9: "Shuffling ... wait"
!   ==> FOR times:=1 TO 75 DO
!   !      fake'move
!   !      move
!   !      //display'board
!   --> ENDFOR times
--> ENDPROC shuffle
    //
==> PROC fake'move
!      init'stick
!   ==> REPEAT
!   !      direction:=RND(1,7)
!   --> UNTIL stick(direction)=TRUE
--> ENDPROC fake'move
    //
==> PROC init'stick
!      empty'square
!      FOR x:=0 TO 8 DO stick(x):=FALSE
!      FOR x:=1 TO 8 STEP 2 DO stick(x):=TRUE
!      IF row=1 THEN stick(5):=FALSE
!      IF row=4 THEN stick(1):=FALSE
!      IF col=1 THEN stick(3):=FALSE
!      IF col=4 THEN stick(7):=FALSE
--> ENDPROC init'stick
    //
==> PROC empty'square
!   ==> FOR rows:=1 TO 4 DO
!   !   ==> FOR cols:=1 TO 4 DO
!   !   !   ==> IF p(rows,cols)=0 THEN
!   !   !   !      row:=rows; col:=cols
!   !   !   !      RETURN
!   !   !   --> ENDIF
!   !   --> ENDFOR cols
!   --> ENDFOR rows
!      STOP "error - no empty square found"
--> ENDPROC empty'square
    //
==> PROC move
!      CASE direction OF
!   ==> WHEN 1 //up
!   !      p(row,col):=p(row+1,col)
!   !      p(row+1,col):=0
!   +-> WHEN 3 //right
!   !      p(row,col):=p(row,col-1)
!   !      p(row,col-1):=0
!   +-> WHEN 5 //down
!   !      p(row,col):=p(row-1,col)
!   !      p(row-1,col):=0
!   +-> WHEN 7 //left
!   !      p(row,col):=p(row,col+1)
!   !      p(row,col+1):=0
!   +-> OTHERWISE
!   !      STOP "error - illegal move"
!   --> ENDCASE
!      move'count:+1
--> ENDPROC move
    //
==> PROC get'move
!      init'stick
!      PRINT AT 24,3: "Use joystick port 2 to move"
!   ==> REPEAT
!   !      joystick(2,direction,dummy)
!   --> UNTIL stick(direction)=TRUE
--> ENDPROC get'move
```
■

Sprite Maker

by Captain COMAL

COMAL Today #8 included a program to turn a character from a font into a sprite. The font'to'sprite procedure listed below is similar, but also lets you choose the size that the character will be. The character is placed in the upper left hand corner of the sprite image. It can be the same size as a normal character, two or three times wider, or twice as tall. There are even more possibilities if you expand the size of the final sprite.

To convert the font character definition into a sprite image, use procedure font'to'sprite. You must include the following 5 parameters: fontset, char, sprite$, xsize, and ysize. Fontset is a number between 0 and 3. 0 and 1 are used to get a character from a user defined font. 2 is used for upper case/ graphics, and 3 is used for upper/ lower case. Char must be in the range of 0 - 255. It matches the screen display codes so use **1** to get an **a**. Sprite$ is the string that will receive the sprite image. It previously must be dimensioned to 64. Xsize is from 1 to 3, 1 is for the normal width. Ysize is from 1 to 2, 2 is for double height.

The get'sprite procedure listed below takes an image on the hi-res graphics screen and converts it into a sprite image. The image may then be moved to another part of the screen and stamped there if desired. Get'sprite has 3 parameters. The first two are the (x,y) coordinates of the upper left hand corner of the portion of the screen which will be converted into a sprite image. The third is the string which will receive the sprite image. It must previously be dimensioned to 64 characters.

Once you have the sprite image, you must use the commands **define, identify, spritepos, spritecolor, and showsprite** before you can see the sprite.

Get'sprite is a little slow; it takes over 5 seconds to get a single sprite image. Could some interested programmer convert this into a package which should be nearly instantaneous?

```
PAGE
DIM s1$ OF 64, s2$ OF 64, s3$ OF 64
USE graphics
USE sprites
graphicscreen(0)
plottext(50,10,"defining sprites from fonts")
font'to'sprite(2,18,s1$,1,1) // R
font'to'sprite(2,2,s2$,1,1) // B
define(1,s1$)
define(2,s2$)
identify(1,1)
identify(2,2)
spritepos(1,160,100)
spritepos(2,168,100)
spritecolor(1,0)
spritecolor(2,0)
spritesize(1,0,0)
spritesize(2,0,0)
showsprite(1)
showsprite(2)
stampsprite(1)
stampsprite(2)
plottext(170,10," from screen")
get'sprite(160,100,s3$) // RB
define(3,s3$)
identify(3,3)
spritepos(3,160,100)
spritecolor(3,0)
spritesize(3,1,1)
showsprite(3)
movesprite(3,50,150,240,0)
plottext(50,10,"  press any key to stop  ")
WHILE KEY$=""0"" DO NULL
//
PROC font'to'sprite(fontset#,char#,REF
 sprite$,xsize,ysize) CLOSED // wrap line
 DIM letter$ OF 8
 IF fontset#<0 OR fontset#>3 THEN REPORT 5
 IF char#<0 OR char#>255 THEN REPORT 5
 IF xsize<1 OR xsize>3 THEN REPORT 5
 IF ysize<1 OR ysize>2 THEN REPORT 5
 // error 5 is value out of range
 USE font
 getcharacter(fontset#,char#,letter$)
 sprite$:=""
```

More ►

Sprite Maker - continued

```
 FOR row:=1 TO 8 DO
   number:=0
   FOR bit'col:=0 TO 7 DO
     bit:=SGN((2^(7-bit'col)) BITAND ORD(letter$(row)))
     FOR count:=1 TO xsize DO number:=number*2+bit
   ENDFOR bit'col
   number:=number*256^(3-xsize)
   FOR c:=1 TO ysize DO
     sprite$:+CHR$(number DIV 65536)
     sprite$:+CHR$((number DIV 256) MOD 256)
     sprite$:+CHR$(number MOD 256)
   ENDFOR c
 ENDFOR row
 FOR count:=1 TO 40 DO sprite$:+CHR$(0)
 sprite$:=sprite$(1:64)
 IF LEN(sprite$)<>64 THEN REPORT 4 // substring error
ENDPROC font'to'sprite
//
PROC get'sprite(x,y,REF sprite$) CLOSED
 // x,y represent the upper left hand corner of the sprite
 USE graphics
 sprite$:=""
 mask:=inq(5) // ignor background color
 FOR row:=0 TO 20 DO
   FOR column:=0 TO 2 DO
     byte:=0
     FOR bit:=0 TO 7 DO
       pixel:=getcolor(x+8*column+bit,y-row)
       IF pixel<>-1 AND pixel<>mask THEN byte:+2^(7-bit)
     ENDFOR bit
     sprite$:+CHR$(byte)
   ENDFOR column
 ENDFOR row
 sprite$:+CHR$(0)
 IF LEN(sprite$)<>64 THEN REPORT 4 // substring error
ENDPROC get'sprite
```

GAME CHALLENGE

The last two articles included a game where pieces of a puzzle were moved around the screen, and procedures to turn a portion of a screen into sprites. These two ideas should work wonderfully together. Here is the challenge: make a new version of the puzzle game which uses sprites as puzzle pieces on the hi-res screen. A hi-res screen could be broken into 16 pieces (4 by 4 square) and each piece converted into a sprite image. **STAMPSPRITE** can be used since there are 16 pieces, but only 8 sprites. The program could include your favorite picture or a routine to let the user choose their own.

Scene Magic

by Richard D Aurland

This program draws on the hi-res graphic screen by stamping sprite shapes. The 30 different shapes listed in a design menu are linked to the program. You can substitute the shapes in the program for ones of your own creation.

Multicolor sprites will not stamp out properly on the graphicscreen. However there is a way around this. Design two or more sprites and stamp one on top of the other. For example, try stamping a red brick wall over a yellow solid block.

The menus in this program are self explanatory, but a few notes may be helpful. A joystick is needed in port 2 to move a sprite around the graphic screen. Pressing the fire button will stamp the sprite onto the screen. Press **m** (from the graphic screen) to go back to the menu to get more sprites or change colors.

Procedure **steal'shapes** has been added to this program to allow you to save the sprite images to disk. The procedure is nearly generic; it may be merged into other programs to obtain their linked sprite images. Just change *shap.magic-* to the name for your sprite images. You can link the images to another program with the **loadshape** command. To use **steal'shapes**, type:

LOAD "scene'magic"
SCAN *// now insert new disk*
steal'shapes

```
PROC steal'shapes CLOSED
  USE sprites
  FOR x:=1 TO 30 DO
    saveshape(x,"shap.magic-"+STR$(x))
  ENDFOR x
ENDPROC steal'shapes
```

Scope Rules

by Richard Bain

What is the output of this program?

```
x:=3
for x:=1 to 5 do null
call'setx
print x
proc call'setx closed
  // import setx // version 2.0 only
  setx
endproc call'setx
proc setx
  x:=4
endproc setx
```

If you said 3, you are right. Of course if you said 6, you are also right. If you said 4, you might be right. Even 5 would be a possible result. Welcome to the world of scope rules.

Many of you may never have heard of scope rules. Most of you will never want to write or run a program like the one presented here. It doesn't matter. *Scope rules* is a term used in computer jargon to explain things like what the output of the above program will be.

Although, the above program is nonsensical, it has an important use. Programmers must know what will happen if the same variable name is used in the main part of the program and also in procedures. How do you find out what will happen? The easiest way is to run the program. However, if you don't know what to look for, even seeing the output may not help much. A better way to find out what will happen **and why** is to look at the documentation of the language. The best source of documentation for COMAL is the *COMAL Handbook*, second edition, by Len Lindsay. Appendix H contains the complete standard defintion of COMAL 80.

Unfortunately, fans of BNF notation can only use this definition to verify that the above example is a valid program. The definition doesn't tell what the program actually does. The fourth note on page 139 about the **FOR** structure may help a little, if you can understand it:

> *(4) The <controlvar> is considered LOCAL to the FOR structure in version 2.0 avoiding possible variable conflicts.*

This actually explains part of why COMAL 2.0 gives a 3 in my example and COMAL 0.14 gives a 6. In COMAL 2.0 (the cartridge and IBM versions) the x in the **FOR** loop is *local.* This means that it is a separate variable from the x that was set to 3. When the **FOR** loop finishes, the x that was originally set to 3 becomes *active* again. In COMAL 0.14, the x in the **FOR** loop is the same x that was set to 3. This is why its value can be changed to 6. (You don't get a 5 because x must be greater than 5 to let the **FOR** loop finish.) If this is confusing to you, there is an easy solution. Never use the value of the control variable (x in our example) of a **FOR** loop after the loop is finished (until you assign a new value to it). The COMAL standard specifies that the control variable is undefined after the loop ends.

So far, I haven't told you why you don't get a 4 in the above example. The short answer is that COMAL has *dynamic scope rules.* Scope rules deal with which identity of a variable is active at any given time. Under *static scope rules*, the identity of a variable is independent of how the program reached the line which uses that variable. Under *dynamic scope rules* the path a program takes to get to a given line is very important.

More ►

Some definitions and examples may help clarify this. By *identity* of a variable, I mean which version of x is active: the x which was set to 3 or the x in the **FOR** loop. X may have several identities, but only one can be active at a time. I do not mean the *value* (3 or 4) of x when I refer to its *identity.*

Now let me explain how the path a program takes to get to a given line can matter. Procedure setx changes the *value* of x to 4, but which *identity* of x is being changed? Setx is an open procedure (it isn't **CLOSED**). Therefore, unless x is being used for the first time, the identity of x within setx must be the identity of x that was active before setx was called. However, x has had two identities: the one that was set to 3 and the one from the **CLOSED** procedure call'setx. (In call'setx, x has an undefined *identity.*)

This is where path and scope rules become important. Under static scope rules, x in setx would have the global identity of x, the one that was first set to 3. This is why the global x could have been set to 4. Under dynamic scope rules, x in setx has the **most recent** identity of x, the one from the **CLOSED** procedure call'setx. This is why the global x is not changed. COMAL 0.14 and 2.0 both have dynamic scope rules, explaining why the number printed is the one from the main program.

As far as I can tell, COMAL's dynamic scope rules are not documented. The only way to find out about them is to run programs like the one I presented here.

Further Reference:

COMAL Handbook, by Len Lindsay ■

Stacks

by Richard Bain

You don't need to know how dynamic scope rules are implemented to be able to use them (see previous article). In fact, COMAL 2.0 and COMAL 0.14 use slightly different implementations to get the same results. Here I will explain how COMAL 2.0 uses two stacks to implement its dynamic scope. Advanced programmers may be interested in this directly. Others may find it helpful in further understanding scope rules.

A running C64 COMAL 2.0 program has five important sections of memory: the tokenized program, the name table, the lower stack, the upper stack, and system pointers. The program starts at $0800. The name table follows immediately after the program. SVARS ($18) points to the start of the name table. The lower stack starts after the name table. SSTACK ($1a) points to the start of the lower stack. STOS ($2d) points to the top (end) of the lower stack. The upper stack starts at ($7fff) and grows towards lower memory. SFREE ($2f) points to the end of the upper stack. There is free memory between the two stacks. Most system pointers: SVARS, SSTACK, STOS, SFREE etc., are in zero page or at $c000.

The program and the name table take a constant amount of memory. To find out how a program is tokenized, see Ian McPhedron's articles in *COMAL Today #9*. Mike Lawrence gives a procedure to print out the name table in the same issue.

The lower stack is for temporary variables and intermediate values from calculations. The variables that go on the lower stack are parameters, nested procedure information, and control variables from **FOR** loops. When a variable is given a new

More ►

identity, its name table entry is updated and the old name table entry is placed on the stack. By changing the name table entry, it is possible to change a variable from one type to another, perhaps from a string to an integer. The value of the original variable is in protected memory, allowing the original variable to be restored when the procedure or **FOR** loop ends. Machine language programmers may have noticed that PSHINT, PUSHRL, and a few other routines also put information on the lower stack.

The upper stack is for permanent information. Global variables go here. Variables created within **CLOSED** procedures also go on the upper stack.

The two stack system is one reason COMAL 2.0 is as fast as it is. Since permanent information is separated from temporary information, there is never a gap in the stack. This is one reason why COMAL doesn't need to perform garbage collections.

You almost have enough information to understand how the stacks relate to dynamic scope rules. There are two more system pointers you need to know about. SCLSD1 ($34) points to the current scope range on the upper stack. SCLSD2 ($36) points to the current scope range on the lower stack. These two pointers are set to point to the end of their stacks every time a **CLOSED** procedure is called. Every time a variable is used in a running program, COMAL first checks to see if it is in a **CLOSED** procedure. If it is not in a **CLOSED** procedure, the identity of the variable must be from the current scope level since it is the only one. No further checks are made before the variable is used.

If COMAL is in a **CLOSED** procedure then there is more than one scope level. COMAL must check to see if the name table is up to date. COMAL compares the address of the variable (from the third and forth byte of the name table entry) with the values of the pointers SCLSD1 and SCLSD2. If the address is outside the range set by the pointers, the variable is from a previous scope and is not current. (Remember that the two stacks grow towards each other from the outside in.) COMAL stores the old name table entry on the upper stack, updates the name table entry, and then places the value of the new variable on the upper stack. Now the address of the variable will be within the range set by the pointers. When the variable is used again, COMAL will know it is current and won't update it a second time. When COMAL leaves the **CLOSED** procedure, it knows to restore SCLSD1, SCLSD2, and the name table entries to their values from before the procedure was called. This is a lot of extra work for **CLOSED** procedures, explaining why they are slower than open procedures.

You need to know one more detail about COMAL 2.0's stacks. Upon entering a procedure, COMAL updates the name table for its parameters, imported variables, and nested procedures before executing any lines of the procedure. **REF** parameters and the **IMPORT** statement are handled in the same way. The old name table entry is stacked and then a bit in the second byte of the name table entry is set indicating the variable is a **REF** variable. The address of the variable is set to point to a two byte pointer which points to the actual value of the variable. The reason for this indirect referencing is that the pointer will be in the range set by SCLSD1 and SCLSD2 even if the value of the variable is not. ■

Trig Art

by Gerald Hobart

The plotting of a mathematical function on a Hi-Res graphic screen is not only a good way to learn about the function, but it also can produce some pretty neat looking graphic screens. The COMAL 2.0 cartridge makes this sort of program very easy to write. The **WINDOW** command in the graphics package is especially convenient.

Without the **WINDOW** command, the (x,y) coordinates of the function must be scaled to fit screen coordinates.This is usually done by the multiplication and addition of certain constants.

With the **WINDOW** command, you leave the function alone. Just redefine the limits of the screen. This is easier to code and provides a program listing which is easier to follow. The syntax for **WINDOW** is:

```
WINDOW(<X min>, <X max>, <Y min>, <Y max>)
```

WINDOW(-10,20,-10,20) would scale the screen so that coordinates of the lower-left corner would be (-10,-10), and those of the upper right would be (20,20).

Forms of the trigonometric functions **SIN(x)** and **COS(x)** are particularly useful for this type of work because they are cyclic and bounded. Because of this you can fill up a good portion of the screen.

Y=SIN(x)*COS(159*x) in the first procedure is plotted for values of x ranging from zero to **PI** in steps of **PI/320**. The design produced is of course largely due to the nature of this function. However, the design is also highly dependent on the size of the plotting interval. Try changing the **STEP** value from **PI/320** to **PI/318** and see what happens. For another variation change **COS(159*x)** to **COS(157*x)**.

Y=SIN(x)*SIN(50*x) is then plotted for values from zero to **3*PI** in steps of .020843. This **STEP** value was arrived at by pure experimentation, and the image produced is highly sensitive to small changes.

If you enjoy this sort of thing, experiment with any of the variables in these procedures and come up with your own graphics. I find it easiest to experiment with one variable at a time. For further variety, change the functions. For example, try multiplying a **LOG** function times a **COS** function and see what you get.

```
USE graphics
border(0)
background(0)
graphicscreen(0)
//
sincos
WHILE KEY$=""0"" DO NULL
sinsin
WHILE KEY$=""0"" DO NULL
textscreen
END
//
PROC sincos
  clear
  pencolor(14)
  window(0,PI,-1.4,1.1)
  moveto(0,0)
  FOR x:=0 TO PI STEP PI/320 DO
    y:=SIN(x)*COS(159*x)
    drawto(x,y)
  ENDFOR x
  pencolor(6)
  textstyle(2,2,0,1)
  plottext(.3,-1.3,"SIN(x)*COS(159*x)")
ENDPROC sincos
//
PROC sinsin
  clear
  pencolor(13)
  window(0,10,-1.4,1.1)
  moveto(0,0)
  FOR x:=0 TO 3*PI STEP .020843 DO
    y:=SIN(x)*SIN(50*x)
    drawto(x,y)
  ENDFOR x
  pencolor(5)
  textstyle(2,2,0,1)
  plottext(.9,-1.3,"SIN(x)*SIN(50*x)")
ENDPROC sinsin
```

Data Base Revisited

by Robert Shingledecker

[The Data Base system from COMAL Today #8 has been improved, and is now unprotected and on Today Disk #14. The following article is adapted from the article in COMAL Today #8.]

After creating a Data Base Program in COMAL 0.14 (see *COMAL Today #6*) I decided to try it again with the more powerful COMAL 2.0. Quite frankly I was impressed with all the power that COMAL 2.0 afforded me in this endeavor. My thanks go to all my friends whose input and sugguestions for improvement help make this program what it is. A special thanks to my nephew Steve Smullen without whom this program may never have been written. For it was after many hours of arguing that we decided on what this program should be. Together we coded, tested, and debuged this system of programs. Also to Larry Phillips whose Screen Wizard program brought to my attention the getscreen and setscreen commands.

Operating Instructions

Disk Preparation

Format a blank disk for use as a data base disk. Type:

PASS "n0:data base,88" *<return>*

Use the *copyfile2.basic* program on *Today Disk #14* (see page 19) to copy these:

```
db'boot,p            db'sort,p
db'define,p          db'squash,p
db'help,p            db'data,s
db'labels,p          db'help.def,s
db'maintenance,p     db'help.lab,s
db'menu,p            db'help.rpt,s
db'report,p          db'name,s
```

Starting the Program

With the data base disk created above mounted in the disk drive type:

RUN "db'boot" *<return>*.

Wait for the main menu to appear:

f1 - Help With System
f2 - Create Database
f3 - Maintain Database
f4 - Purge Database
f5 - Report Database
f6 - Database Labels
f7 - List Database Directory
f8 - End program

Help With System: f1 from Main Menu

This option takes you to a menu of help screens. Choose this option and read each help screen the first time you run the program.

Create a Database: f2 from Main Menu

Before you begin this option you should plan ahead. Think about what you want to capture. Think about how the data will have to be sorted. Sorting is on a field level. You can have up to twenty fields per record. Records may contain up to 254 characters. The field name plus the field length must be able to be displayed on one line, i.e. maximum 40 characters. Pre-planning will yield the most useful dictionary thus a more useful data base.

First enter the name of this data base. Do not use *'dict* or *'data* as these are reserved suffixes used by the program. Use simple one word names.

Next you will be given a blank screen to design your own input form. Use only the

More ►

cursor keys to position the input prompts and use the special characters [ ] to indicate the field lengths. Only the first twenty lines may be used. When you are satisfied with your input form, press f7.

The program will next verify your format and create a data dictionary. If no errors occured then enter the approximate size of the data base. That is how many records. Be realistic! This is **not** an IBM mainframe! Keep the size small, less than 100 records works best. The system will handle up to 400 records but will take a long time to sort! The limit of 400 records is based on the number of free blocks with a maximum record size.

Now be patient as the system will set up the disk files for the dictionary and the data portions of your data base.

Maintain Database: f3 from Main Menu

Add records: f1 from Maintain Menu

You will be given the input form that you designed. Use the delete key to correct mistakes within a field. After all fields have been entered you will be prompted to accept the record or re-input the entire record. If you are entering a record with many fields and discover an error it might be best to accept this record and change it later with the **change** command. The **change** command allows easy changes by field. Finally when you are finished adding records simply send a blank form to return to the Maintain Database Menu.

Change records: f2 from Maintain Menu

Use this option to change any or all fields of an existing record. Once you select this option you will be prompted for the select method desired. Once a select option is chosen the system will try to find the desired record(s). Once found the record contents will be displayed. Enter y to change the record. Then each prompt and input field will be displayed. If you **do not** want to change this particular field simply press *<return>* key. The original field contents will appear and the next field prompt and input field will appear. By simply pressing the *<return>* key you can *tab* to the desired field to change. When all fields have been displayed the program will respond with *Change record? (y,n)* for a final approval. If you select **n** you can still recall the original field contents. When you select y all fields will be updated.

Display records: f3 from Maintain Menu

Use this together with the select options for a powerful combination of displaying information. Simply press *<space>* to view each record that matches the chosen select option.

Delete records: f5 from Maintain Menu

This option will delete records from the data base. Once this option is selected you will be prompted for the select option. For each record that matches the select option chosen, the record will be displayed and a final prompt to delete it will be given.

Use Index: f7 from Maintain Menu

This option is only available after initial use of the Maintain Database program (the second time you run the program). Everytime you use the Maintain section to add, change, or delete records then upon exiting from this section *db'sort* and or *db'squash* is automatically

More ►

called to create the index files for each defined field. This version of the sort is **much faster** than the in-place disk sort of the COMAL 0.14 version. Instead of an actual record sort it creates index files.

Select Options Menu
common submenu to maintain options menu

f1 - Complete Search
f3 - Process All Records
f7 - Return to Menu

Complete search: f1 from Select Menu

Enter the characters that should reside in the records you desire. The less you specify the more records that are likely to contain them and thus be selected. The more characters you specify the less records will be selected. Each record that contains the specified string regardless in what field will be selected for processing by the previously specified mode, i.e., changing, deleting, displaying, or printing. However if you have specified the use of an **index** then the search is extremely fast and will include only the field that the index is defined for.

Process all records: f3 from Select Menu

This option will select **all** records in the file for whatever mode of operation was chosen: changing, deleting, or, displaying.

Return to menu: f7 from Select Menu

F7 will return you to the Maintain Menu.

Purge Database: f4 from Main Menu

Enter the data base name to be deleted from the system.

Report Database: f5 from Main Menu

You may select which fields are to be printed and in which order they are to appear. Optionally you can use any index to have them sorted by that field. No more waiting for a sort when you want a report. You will also be prompted for a title. The 80 columns can then be sent to the printer or the screen. See help screen for details.

Database Labels: f6 from Main Menu

Again from the input form you may specify up to nine displayable fields accross 5 printable lines. The fields may be in any order and you can combine several fields per label line. The labels can be sorted by indicating which index to use. Again the index files are there for immediate use. See help screen for details. *[A printer is required for this option.]*

List Database directory: f7 from Main Menu

This option will display all data bases residing on the disk.

Well, that's it. If anyone cares to send comments, wish-list, and or bug reports, they can write me at:

Robert Shingledecker
12555 Euclid Street #27
Garden Grove, California 92640

[The data base system is self prompting and help screens] ■

Now Available Through Aquarian Software

Gold Disk Series

Volumes 1 through 11 Now Available!!!

Volume 11 Features a C-64 Assembler

Each Disk Contains:

- The Monthly Feature Program
- Programming Tutorials
- High Quality Games
- And Much More

Gold Disk Series for 128
Coming Soon!

Only $14.95 Per Disk*

* Plus Shipping and Handling

The Cataloger

The Ultimate Disk Cataloging System for the 64!

Features of The Cataloger V3.5A Include:

- ★ Loads directly from the disk itself.
- ★ Ability to change name of entry.
- ★ Fast — Uses relative files exclusively
- ★ Search, Sort and Print by any of 12 fields.
- ★ 1100-program (or disk) capacity per data disk.
- ★ All machine language.
- ★ Menu driven — very easy to use.
- ★ Works with one or two drives.

Only $24.95

BobsTerm Pro

The Ultimate Terminal Software !

Upload / Download Supports Punter, X-Modem, XON / XOFF, DC1 / DC2, and Much More!

28.5 Byte Buffer with unmatched editing abilities

- User Adjustable Parameters
- 10 Custom Character Sets
- Unlimitied Phone Book Storage
- Programmable Macro Command Strings

Only $59.95

MATRIX — NOW AVAILABLE!!

The Indispensable C-128 Utility / Starter Kit !

Use **dozens** of 128 features in the 64 mode:

- Numeric Key Pad
- Cursor Keys
- 80-Column RGB Output
- Many Other Special Function keys

One-Key Functions Include:

- **2 Megahertz "Fast Mode"**
- **One-Key Screen Dumps**
- **Full-Featured DOS Utility Menu**

Other Features Include:

- Fast Loading
- Fast Copy For The 1571!
- Relocatable In Memory
- 100% Transparent to BASIC

Available Now For Only $59.95

Graphic Screen Exporter

A Universal Graphics Converter !

Converts Anything to Anything — Including:

Koala Pad	Doodle
Flexidraw	Print Shop
COMAL	Paint Magic
CAD GEM	Micron Eye

And Many Many More ! !

The Most Versatile Graphics Utility Ever Released for the Commodore 64 !

Only $29.95

ALSO AVAILABLE:

OmiTerm......$19.95
Full-Feature Terminal at an Affordable Price!
Turbo Calc/64......$17.95
A great spreadsheet at an Unbelievable Price!
Tax Computation......$29.95
The friendliest tax package on the market.
Guitar Master......$49.95
A comprehensive musical instruction package
Fast Boot!......$14.95
Mike J. Henry's Fast Loader for 1541/MSD
Thriller Collection......$24.95
Seven intricate text adventures on one disk

Call or Write for Full Catalog !

CAD-GEM

Computer Assisted Design Graphic Element Manipulation

A Wire Frame CAD system for the C64 !
Input from Joystick, Track Ball, Light Pen or Graphics Tablet
360 Degree Rotation in .1 Degree Increments
Scaling on a 64K x 64K, 2048 Mega-Bit Virtual Screen
Independent Manipulation of 400 Objects (Points or Lines)

You must see CAD GEM to believe it!
Demo Disk Available for $3.00

$89.95

MODEM MASTER

The Friendliest Commodore BBS Available

- Works with 1541 or MSD Dual Drive
- 300 / 1200 Baud Operation
- New Punter File Transfer Protocol
- Sub-Directories for File Transfer
- 250 User Capacity
- Accurate Clock / Calendar
- Printer Output
- Information Files
- "Old" E-Mail Deleted After One Week
- Set Up In Only 10 Minutes !

Only $29.95

Total Software Development System

by Kevin Pickell
Now Available in the States !

Assembler/Editor — fast load, get, log and loadat; adds 38 new commands; full macro instructions; allows 13-character labels; assembles to and from disk
Sprite Editor — 256 sprites in memory, view 64 at same time, works with keyboard, joystick or trackball, animates sprites during design
Unassembler — create source code from any ML program
Sound Editor — create interrupt-driven sound effects
Character Editor — edit all characters. Screens to 255x64. Hi-res & Multi-color Character Sets
TSDS automatically includes sprites, characters, mattes and sound effects into source code!

Only $39.95
128 Version Coming Soon !

Aquarian Software
P.O. Box 22184
Portland, OR 97222

VISA

To order, Call: (503) 654-2641
VISA & MasterCard Accepted

MasterCard

Add 3.00 S & H Per Order
(Add Additional $2.00 for COD)
Canadian Orders Add 10.00 S&H
Allow 3-4 Weeks For Delivery

Write or Call for Full Catalog — Dealer Inquiries Welcome !

Inventory Programs

by L. W. Zabel

The three programs, *inventory-1*, *inventory-2* and *inventory-3* calculate minimum cost strategies for controlling an inventory.

Most large businesses have sophisticated programs which they use to keep their purchasing and inventory costs at a minimum. This is much less likely to be the case for small businesses where purchases are usually made solely on the basis of intuition.

The first program should be used when the purchasing lead time, ie. the time between the placement of an order and its delivery, is known and fixed. The other variable which must be known and fixed is the demand or the rate at which items are removed from inventory. For the purposes of these programs, the unit of time is the day. When running the programs, any other unit of time may be used as long as its use is consistent throughout the data entry.

The equation for the total cost was developed and differentiated with respect to the quantity to be ordered. The resultant equation is set to zero and evaluated. The output is in terms of the purchasing quantity and the length of time between purchases.

One of the input items is the cost of a shortage. If the user cannot tolerate any shortage at any time, he should enter a large number when shortage cost is requested. Otherwise he should enter his best estimate of the cost of a single shortage in a single day. This cost should include, not only the direct loss of a sale, but the loss in customer good will as well.

Frequently when a purchase order is issued, there are discounts for quantity purchases. If so, the program will request the necessary data so that they may be included in the minimum cost calculations.

The second program covers a much more complicated situation. That is the condition where both the lead time and demand are variable. It would be a very rare condition where a mathematical equation could be written which would describe the relationships between these variables and the total cost. Hence one is required to use probability distributions. The process can be roughly described as follows: Data on lead times for past purchases are entered as well as historical data on demand per day. From these data, probability distributions are generated. These distributions are then used in a Monte Carlo simulation to determine the minimum cost conditions. It should be apparent that the conditions are not static and that the future will not exactly fit the past. It would be wise for the user to rerun the program every few months in order to maintain his costs near the true minimum.

As written, the program limits the historical data to 200 items each for lead time and demand. It also limits the maximum demand per day to 100 items and the lead times to 100 days. These limits can easily be changed by changing the dimension statements on lines 240 and 250. Monte Carlo simulations require many iterations to be valid. The major iteration loop is set at 1000, which is about the minimum that should be used. This figure is a compromise between precision and running time. The running time depends on the amount and nature of the historical data used. The total number

More ►

of iterations in the simulation will run from a minimum of about 4000 to a maximum of roughly 100,000. Thus the running time could be several minutes.

As with program 1, the Monte Carlo program will accept quantity purchase discounts. There is one limitation to the data entry that should be mentioned. If either the lead time or the demand is fixed, you cannot enter a single figure and expect the program to run. This peculiarity can be circumvented by entering, for example, twenty lead times that are all the same and one that is one day different. This will result in a very narrow frequency distribution which will cause a barely detectable error in the minumum cost figure since the cost minima are always quite broad valleys.

In many circumstances, multiple items are ordered at the same time, from the same supplier. Since only one purchase order is written for multiple items, the purchasing cost is distributed between the various items. When doing this, only one of the items can be ordered under minimum cost conditions. The distribution of the purchasing cost can be made to give an over-all cost minimum. This requires multiple cost calculations with a different purchasing cost distribution for each iteration. Fortunately, the system converges quickly so that only 10 to 20 iterations are required. The program has been set at 20 iterations.

In order to use Inventory-3, one must first use either Inventory-1 or Inventory-2 to determine the minimum cost strategy for each item assuming that each item will bear all the cost of purchasing. The output from these calculations for each item is then the source of information to be used in the final calculation of the single purchase order case. The output from Inventory-3 gives the reorder period, the quantity of each item to be ordered, the cost per day attributable to each item and the total cost per day.

I hope that these programs will be of some use to those of you who are operating small businesses and to those others who may be interested in the techniques of the Monte Carlo simulation.

Inventory program #1

```
init
title
data'input
disp
END "That is all...Good Bye"
//
PROC init
  USE system
  textcolors(1,1,0)
  DIM z$ OF 10, a$ OF 1
  DIM number(10), cost(10), tc(10)
  z$:="$###.##"
ENDPROC init
//
PROC title
  PAGE
  PRINT TAB(15),"INVENTORY-1"
  PRINT TAB(15),"L. W. Zabel"
  PRINT
  PRINT "This program will calculate the minimum"
  PRINT "cost purchasing strategy for the case"
  PRINT "where the demand is known and fixed."
  PRINT "Where discounts are available for"
  PRINT "quantity purchases, the program"
  PRINT "will accept these additional data."
  PRINT "When shortages are not permitted,"
  PRINT "enter a very large number for the cost"
  PRINT "when requested by the program."
  PRINT
  INPUT AT 0,0,0: "Press RETURN to continue: ": b$
ENDPROC title
//
PROC data'input
  PAGE
  INPUT "Purchasing lead time in days? ": tim
  INPUT "Demand in unit per day? ": dem
  INPUT "Cost per unit in $? ": co'unit
  INPUT "Purchase order cost in $? ": co'pur
  INPUT "Storage cost per unit per day in $? ": co'inv
  INPUT "Shortage cost per unit per day in $? ": co'sh
  PRINT
  PRINT "Is there a discount for quantity"
```

More ►

```
  INPUT "orders? (y/n) ": a$
  PRINT
  IF a$="y" OR a$="Y" THEN
    PRINT "Enter 0 as number and cost after"
    PRINT "final entry."
    n:=1
    number(1):=1
    REPEAT
      PRINT
      PRINT "Quantity break point number:";n
      INPUT "Breakpoint quantity? ": number(n)
      INPUT "Cost above breakpoint? ": cost(n)
      n:=n+1
    UNTIL number(n-1)=0
    calc
    minimize
  ELSE
    calc
  ENDIF
ENDPROC data'input
//
PROC calc
  quant:=SQR(2*co'pur*dem/co'inv+2*co'pur*dem/co'sh)
  quant:=INT(quant+.5)
  lev:=dem*tim-co'inv*quant/(co'inv+co'sh)
  lev:=INT(lev+.5)
  tot'cost:=co'unit*dem+SQR(2*co'pur*co'inv*co'sh*dem/(
  co'inv+co'sh)) // wrap line
  per:=quant/dem
  per:=INT(per*100)/100
ENDPROC calc
//
PROC minimize
  FOR i:=1 TO n-2 DO
    tc(i):=cost(i)*dem+co'pur*dem/number(i)+co'inv*num
    ber(i)/2 // wrap line
  ENDFOR i
  FOR i:=1 TO n-2 DO
    IF tot'cost>tc(i) THEN
      tot'cost:=tc(i)
      quant:=number(i)
      per:=quant/dem
      co'unit:=cost(i)
    ENDIF
  ENDFOR i
ENDPROC minimize
//
PROC disp
  PAGE
  print'out
  INPUT "Do you want a hard copy? (y/n)?": a$
  IF a$="y" OR a$="Y" THEN
    SELECT OUTPUT "lp:"
    print'out
    SELECT OUTPUT "ds:"
  ENDIF
ENDPROC disp
//
PROC print'out
  print "Purchasing lead time is:";tim;"days."
  print
  print "Demand is:";dem;"units per day."
  print
  print using "Cost per unit ="+SPC$(17)+z$: co'unit
  print
  print using "Purchasing cost is ="+SPC$(12)+z$: co'pur
  print
  print using "Storage cost per unit per day = "+z$: co'inv
  print
  print using "Shortage cost per unit per day ="+z$: co'sh
  print
  print using "The total cost per day ="+SPC$(8)+z$: tot'cost
  print
  print "Quantity to be ordered is";quant;"units"
  print
  print "The period between purchases is";per;"days."
  print
ENDPROC print'out
```

Inventory program #2

```
DIM z$ OF 10, a$ OF 1, b$ OF 1, number(10), cost(10)
DIM tc(10), units(200), tim(200), freq'u(100), freq't(100)
z$:="$###.##"
//
PAGE
PRINT TAB(15),"INVENTORY-2"
PRINT TAB(15),"L. W. Zabel"
PRINT
PRINT "This program will calculate the minumum"
PRINT "cost purchasing strategy for the case"
PRINT "where the demand and purchasing lead"
PRINT "time are variable and probabalistic."
PRINT "Historical data are used to determine"
PRINT "the frequency distributions of these"
PRINT "variables."
PRINT
INPUT "Press RETURN to continue: ": a$
data'input
//
PROC data'input
  PAGE
  INPUT "Cost per unit in $? ": co'unit
  INPUT "Purchase order cost in $? ": co'pur
  INPUT "Storage cost per unit per day in $? ": co'inv
  INPUT "Shortage cost per unit per day in $? ": co'sh
  PRINT "Is there a discount for quantity"
  INPUT "orders? (y/n)? ": a$
  IF a$ IN "yY" THEN
    n:=1
    number(1):=1
    REPEAT
      PRINT "Enter 0 as number and cost after"
      PRINT "final entry."
      PRINT "Quantity break point number:";n
      INPUT "Breakpoint quantity? ": number(n)
      INPUT "Cost above breakpoint? ": cost(n)
      n:=n+1
    UNTIL number(n-1)=0
  ENDIF
  IF a$ IN "yY" THEN
    history
    min'max
    freq'dist
```

More ►

```
   prob'dist
   simulate
   av
   minimize
   disp
  ELSE
   history
   min'max
   freq'dist
   prob'dist
   simulate
   av
   disp
  ENDIF
ENDPROC data'input
//
PROC history
  PAGE
  PRINT "Enter historical data.  Each entry is"
  PRINT "the demand for one day."
  PRINT "To stop, enter a negative number"
  l:=1
  REPEAT
    PRINT "Day number:";l
    INPUT "Demand (number of units)?": units(l)
    l:=l+1
  UNTIL units(l-1)<0
  PAGE
  PRINT "Enter historical data. Each entry is"
  PRINT "the purchasing lead time experienced"
  PRINT "for one order."
  PRINT "To stop, enter a negative number."
  m:=1
  REPEAT
    PRINT "Order number:";m
    INPUT "Lead time in days? ": tim(m)
    m:=m+1
  UNTIL tim(m-1)<0
ENDPROC history
//
PROC min'max
  min'units:=10000
  max'units:=0
  min'tim:=10000
  max'tim:=0
  FOR i:=1 TO l-2 DO
    IF min'units>units(i) THEN min'units:=units(i)
    IF max'units<units(i) THEN max'units:=units(i)
  ENDFOR i
  FOR i:=1 TO m-2 DO
    IF min'tim>tim(i) THEN min'tim:=tim(i)
    IF max'tim<tim(i) THEN max'tim:=tim(i)
  ENDFOR i
ENDPROC min'max
//
PROC freq'dist
  FOR j:=min'units+1 TO max'units+1 DO
    freq'u(j):=0
    FOR i:=1 TO l-2 DO
      IF j-1=units(i) THEN freq'u(j):=freq'u(j)+1
    ENDFOR i
  ENDFOR j
  FOR j:=min'tim+1 TO max'tim+1 DO
    freq't(j):=0
    FOR i:=1 TO m-2 DO
      IF j-1=tim(i) THEN freq't(j):=freq't(j)+1
    ENDFOR i
  ENDFOR j
ENDPROC freq'dist
//
PROC prob'dist
  FOR i:=min'units+1 TO max'units+1 DO
    freq'u(i+1):=freq'u(i)+freq'u(i+1)
  ENDFOR i
  FOR i:=min'units+1 TO max'units+1 DO
    freq'u(i):=freq'u(i)/freq'u(max'units+1)
  ENDFOR i
  FOR i:=min'tim+1 TO max'tim+1 DO
    freq't(i+1):=freq't(i)+freq't(i+1)
  ENDFOR i
  FOR i:=min'tim+1 TO max'tim+1 DO
    freq't(i):=freq't(i)/freq't(max'tim+1)
  ENDFOR i
ENDPROC prob'dist
//
PROC simulate
  tot'dem:=0
  tot'quant:=0
  tot'lev:=0
  tot'lead'time:=0
  total'cost:=0
  PAGE
  PRINT AT 12,4: "Please wait...I'm doing my best!"
  FOR j:=1 TO 1000 DO
    dem:=0
    RANDOMIZE
    x:=RND
    i:=max'units+1
    REPEAT
      dem:=i-1
      i:=i-1
    UNTIL x>=freq'u(i) OR i=1
    lead'time:=0
    RANDOMIZE
    y:=RND
    k:=max'tim+1
    REPEAT
      lead'time:=k-1
      k:=k-1
    UNTIL y>=freq't(k) OR k=1
    calculate
    total
  ENDFOR j
ENDPROC simulate
//
PROC calculate
  quant:=SQR(2*co'pur*dem/co'inv+2*co'pur*dem/co'sh)
  lev:=dem*lead'time-co'inv*quant/(co'inv+co'sh)
  w:=SQR(2*co'pur*co'inv*co'sh*dem/(co'inv+co'sh))
  tot'cost:=co'unit*dem+w
ENDPROC calculate
//
PROC av
  av'quant:=INT(tot'quant/1000+.5)
  av'dem:=INT(tot'dem/1000+.5)
  total'cost:=INT(total'cost/1000+.5)
```

More ►

```
  av'lead'time:=INT(tot'lead'time/1000+.5)
ENDPROC av
//
PROC total
  tot'dem:=tot'dem+dem
  tot'lead'time:=tot'lead'time+lead'time
  tot'quant:=tot'quant+quant
  tot'lev:=tot'lev+lev
  total'cost:=total'cost+tot'cost
ENDPROC total
//
PROC minimize
  FOR i:=1 TO n-2 DO
    tc(i):=cost(i)*av'dem+co'pur*av'dem/number(i)+co'inv*
    number(i)/2 // wrap line
  ENDFOR i
  FOR i:=1 TO n-2 DO
    IF total'cost>tc(i) THEN
      total'cost:=tc(i)
      av'quant:=number(i)
      co'unit:=cost(i)
    ENDIF
  ENDFOR i
ENDPROC minimize
//
PROC disp
  PAGE
  print'out
  INPUT "Do you want a hard copy? (y/n) ": a$
  IF a$ IN "yY" THEN
    SELECT OUTPUT "lp:"
    print'out
    SELECT OUTPUT "ds:"
  ELSE
    PRINT "That is all...Good Bye"
    END ""
  ENDIF
ENDPROC disp
//
PROC print'out
  print "Expected purchasing lead time is";av'lead'time;"days."
  print
  print "Expected demand is";av'dem;"units per day."
  print
  print using "Cost per unit is"+spc$(25)+z$: co'unit
  print
  print using "Purchasing cost is"+spc$(23)+z$: co'pur
  print
  print using "Storage cost per unit per day is"+spc$(9)+z$:
  co'inv // wrap line
  print
  print using "Shortage cost per unit per day is"+spc$(8)+z$:
  co'sh // wrap line
  print
  print "Expected order quantity is";av'quant;"units."
  print
  period:=INT(av'quant*100/av'dem)/100
  print "Expected period between purchases is";period;"days."
  print
  print using "Expected total cost per day is"+spc$(11)+z$:
  total'cost // wrap line
  print
ENDPROC print'out
```

Inventory program #3

```
DIM name$(20) OF 20, pur'cost(20), co'sh(20), tot'cost(20)
DIM quant(20), period(20), co'unit(20), dem(20), co'inv(20)
DIM a$ OF 1
//
PAGE
PRINT TAB(15),"INVENTORY-3"
PRINT TAB(15),"L. W. Zabel"
PRINT
PRINT "This program calculates the minimum "
PRINT "cost strategy for the purchase of"
PRINT "multiple items on the same purchase"
PRINT "order from the same supplier.  Program"
PRINT "Inventory-1 or Inventory-2 will have"
PRINT "to be run for each item in order to"
PRINT "obtain the input data for Inventory-3."
PRINT
INPUT "Press RETURN continue: ": a$
data'input
iterate
select'max
calculate'cost
display
END "That is all...Good Bye"
//
PROC data'input
  PAGE
  INPUT "Purchasing lead time? ": lead'time
  n:=1
  INPUT "Purchasing cost? ": co'pur
  REPEAT
    INPUT "Product name? ": name$(n)
    INPUT "Minimum cost quantity? ": quant(n)
    INPUT "Minimum cost period? ": period(n)
    INPUT "Unit cost? ": co'unit(n)
    INPUT "Demand per day? ": dem(n)
    INPUT "Inventory cost per unit per day? ": co'inv(n)
    INPUT "Shortage cost per unit per day? ": co'sh(n)
    PRINT
    INPUT "Another item?": a$
    n:=n+1
  UNTIL a$ IN "nN"
ENDPROC data'input
//
PROC proportion'pur'costs
  tot:=0
  FOR i:=1 TO n-1 DO
    tot:=tot+quant(i)*co'unit(i)
  ENDFOR i
  FOR i:=1 TO n-1 DO
    pur'cost(i):=quant(i)*co'unit(i)*co'pur/tot
  ENDFOR i
ENDPROC proportion'pur'costs
//
PROC calc'quantity
  FOR i:=1 TO n-1 DO
    q:=2*pur'cost(i)*dem(i)/co'inv(i)
    r:=2*pur'cost(i)*dem(i)/co'sh(i)
    quant(i):=SQR(q+r)
    period(i):=quant(i)/dem(i)
  ENDFOR i
ENDPROC calc'quantity
//
PROC select'max
```

More ►

Inventory Programs - continued

```
  per:=0
  FOR i:=1 TO n-1 DO
    IF per<=period(i) THEN per:=period(i)
  ENDFOR i
  per:=INT(per*100)/100
  FOR i:=1 TO n-1 DO
    quant(i):=per*dem(i)
  ENDFOR i
ENDPROC select'max
//
PROC iterate
  j:=1
  REPEAT
    proportion'pur'costs
    calc'quantity
    j:=j+1
  UNTIL j=21
ENDPROC iterate
//
PROC calculate'cost
  FOR i:=1 TO n-1 DO
    s:=co'unit(i)*dem(i)+pur'cost(i)*dem(i)/quant(i)
    t:=co'inv(i)*co'sh(i)*quant(i)/(2*(co'inv(i)+co'sh(i)))
    tot'cost(i):=s+t
  ENDFOR i
  total'cost:=0
  FOR i:=1 TO n-1 DO
    total'cost:=total'cost+tot'cost(i)
  ENDFOR i
ENDPROC calculate'cost
//
PROC display
  PAGE
  FOR i:=1 TO n-1 DO
    quant(i):=INT(quant(i)+.5)
  ENDFOR i
  print'out
  INPUT "Do you want a hard copy? (y/n)": a$
  IF a$ IN "yY" THEN
    SELECT OUTPUT "lp:"
    print'out
    SELECT OUTPUT "ds:"
  ENDIF
ENDPROC display
//
PROC print'out
  PRINT "Order each";per;"days"
  PRINT
  FOR i:=1 TO n-1 DO
    PRINT "Order";quant(i);name$(i)
    PRINT
    PRINT "Cost attributable to";name$(i);"is";
    PRINT USING "$###.##": tot'cost(i);
    PRINT "per day."
    PRINT
    PRINT
  ENDFOR i
  PRINT USING "The total cost is $###.##": total'cost;
  PRINT "per day."
  PRINT
ENDPROC print'out
```
■

Define Keys

by Jack Baldridge

I recently wanted to write a COMAL 0.14 program (something I had done only once or twice since I got my 2.0 cartridge, to be honest about it). I suddenly realized that I couldn't use function keys to enter common editing commands. I then took a recess for a few hours while I adapted a program I had once written to the COMAL environment. *Define'keys*, on *Today Disk #14*, is the result.

With it, you can assign commands to twelve function key combinations; these are the four unshifted function keys, the shifted keys, and the keys with the Commodore logo key. The machine language program is 272 bytes long and starts at 49800. I placed it where I did so it won't interfere with errors in RAM. The function key definitions are placed in data statements. To activate the function keys, **RUN** the program.

With the program in operation, you are restricted to the first nine sprites. In addition, the use of graphics mode causes complications, since function keys F1, F3, and F5 will each have two definitions. I advise deactivating the keys with SYS 49803 if you will be using graphics. To reactive them, type SYS 49800.

The COMAL 0.14 program has a loader for both numeric and alpha data. Alpha strings up to eleven characters may be assigned; each string is twelve characters long, but '@' must end each string, since the program uses it as a delimiter.

Further Reference:

The Enhancer, COMAL Today #12, page 4 ■

COMAL Outside the USA

GERMANY

COMAL Users Group Bielefeld
c/o K D Polloczek
Paulusstr. 19
D-4800 Bielfeld 1
West Germany
phone 0521-887915

COMAL User Gruppe
Christiane Canisius
Freiheitstr. 30
4000 Dusseldorf 12
West Deutchland Germany
phone 0211-279336

COMALGRUPPE-Deutschland
D. Belz
2270 Utersum/ Fohr
West Germany
phone 04683/500
modem 04683/554
Publish: COMAL-News

DENMARK

COMAL Bruger Gruppe
Mindegade 42
8700 Horsens
Denmark
phone 05-621567

COMAL Today DK
Postbox 1222
DK 2300 Copenhagen S
Denmark
Distributes: COMAL Today

SWEDEN

Ake Fredriksson
Comalklubben I Sverige
Gustavsbergsgatan 8
S-431 37 Molndal
Sweden
Publish: COMAL-Nyheterna

HOLLAND

Dick Klingens
Dutch COMAL Users Group
Violeir 3
2925 TE
Krimpen ad IJssel
Holland, the Netherlands
Publish: COMAL-80 info

ENGLAND

Brian Grainger
ICPUG COMAL SIG
73 Minehead Way
Stevenage, Herts
England SG1 2HZ
Phone +44 438 727925
Publish: ICPUG Newsletter

SCOTLAND

COMAL Users Group (*possible*)
Angus Quinn
45 Fotheringay Road
Pollokshields
Glasgow, G41 4NN Scotland

AUSTRALIA

Wayne Vovil
COMAL Users Group Australia
PO Box 899
Woden, ACT 2606
Australia

CANADA

Donald Dalley
TPUG COMAL SIG
423 - 2770 Jane St.
Downsview, ONT Canada M3N 2J1
Publish: TPUG Magazine

See also international vendor addresses in COMAL Today #11, page 10-12 ■

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 more 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 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 our new **READ & RUN** series disks. ■

How to Type in COMAL 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

While entering a program, use unshifted letters. If letters are capitalized in the listing it does not mean to use SHIFT with those letters. They are capitalized merely to be easy to read. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures. You **DO** have to type a space between keywords in the program.

Long program lines: If a complete program line will not fit on one line, we will continue it onto the next line and add *//wrap* at the end. You must type it as one continuous line. Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz 0123456789
'] [<backslash> _

The <left arrow> key in the upper left corner of the keyboard is valid. COMAL 2.0 converts it into an underline. If you see an underline in a program listing, type the <left arrow> key. The C64 and C128 computers use a <british pound> in place of the <backslash>. ■

Disk Directories For Disk Sleeves

Today Disk #13 - Front

			87 Files	2 Blocks Free:	contributors:
boot c64 comal	circle/demo	string'art	vocab'sample.dat	~~~~~~~~~~~~~~~~	Phyrne Bacon
comal 64 0.14	cubes	voc'create'data	~~~~~~~~~~~~~~~~	1000primes.basic	Jack Baldridge
comalerrors	dir'probe'boot	voc'fix'it	~ functions ~	byte'calc.basic	Ed Bolton
ml.sizzle	directory'probe	voc'instructions	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	Bert Denaci
hi	disk'editor	voc'screen'color	file'exists.func	~for more info ~	Henry Farkas
menu	disk'editor'boot	vocabulary	ti.func	~ send sase to ~	Rodger Godsey-Bell
~~	dynamic'new	yes'no'maybe	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	Oren Hasson
~comal programs~	dynamic'new2	zoo'match'game	~ procedures ~	~ comal users ~	Jeff Herdlicka
~~	error'message	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ group, usa ~	Sol Katz
1000primes	keywords1	~ data files ~	call.proc	~ 6041 monona ~	Len Lindsay
1520-3d'airplane	keywords2	~~~~~~~~~~~~~~~~	expand'ram.proc	~ madison, wi ~	Greg Mavko
1520-3d'object	print'directory	~ do not load ~	inkey.proc	~ 53716 ~	Terry Mills
1520-shocking	quick'sprites	~~~~~~~~~~~~~~~~	load'errors.proc	~~~~~~~~~~~~~~~~	Kevin Quiggle
3d-view'airplane	samaritan	comal'zoo.dat	~~~~~~~~~~~~~~~~	~(608)222-4432 ~	Robert Ross
3d-view'object	sea'sick	comalhelp.txt	~basic programs~	~~~~~~~~~~~~~~~~	David Stidolph
basic2comal-p1	shocking	programs.dat	~~~~~~~~~~~~~~~~		David Zavitz
basic2comal-p2	sig'digits	samaritan.dat	~ do not load ~		
byte'calc-5/85	slot'machine	slot'sprites.dat	~ from comal ~		

Today Disk #13 - Back

			105 Files	5 Blocks Free:	contributors:
hi	kastle	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ source code ~	Richard Aurland
~~	sets/demo	~copyright 1985~	~ functions ~	~~~~~~~~~~~~~~~~	Richard Bain
~comal programs~	shocking	~ j. blake ~	~~~~~~~~~~~~~~~~	src.version	Ed Bolton
~~	sig'digits	~ lambert ~	func.blocks'free	~~~~~~~~~~~~~~~~	Bert Denaci
1000primes	vdc'editor.pop	~~~~~~~~~~~~~~~~	func.c128'in'mem	~ packages ~	Oren Hasson
1520-3d'airplane	wheel'of'fortune	~ all rights ~	func.file'exists	~~~~~~~~~~~~~~~~	Bob Hoerter
1520-3d'object	zoo'match'game	~ reserved ~	func.superchip	pkg.c128	Dick Klingens
1520-draw'head	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	pkg.code'doctor	Steve Kortendick
3d-view'airplane	~character sets~	font.boone	~ procedures ~	pkg.quake	J Lambert
3d-view'object	~~~~~~~~~~~~~~~~	font.chicago	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	Len Lindsay
byte'calc-5/85	~ do not load ~	font.newyork	proc.center	~for more info ~	Kevin Quiggle
chip.1000primes	~~~~~~~~~~~~~~~~	font.vilas	proc.copyscreen	~ send sase to ~	Joel Rea
chip.demochip	~ these fonts ~	set.boone	proc.xactcopy	~~~~~~~~~~~~~~~~	Patrick Roye
chip.primefactor	~ taken with ~	set.chicago	~~~~~~~~~~~~~~~~	~ comal users ~	David Stidolph
chip.testchip	~ permission ~	set.newyork	~ batch file ~	~ group, usa ~	
code'doctor/dem2	~ from ~	set.vilas	~~~~~~~~~~~~~~~~	~ 6041 monona ~	
code'doctor/demo	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ use ~	~ madison, wi ~	
crazy'clock	~ /speedpak/ ~	~ data file ~	~ select input ~	~ 53716 ~	
cubes	~ pob 22022 ~	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	
easyreader	~ greensboro, ~	dat.comal'zoo	bat.cleanup	~(608)222-4432 ~	
encrypt'file	~ nc 27420 ~	hrg.kastle	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	

Today Disk #14 - Front

			81 Files	6 Blocks Free:	Phyrne Bacon
boot c64 comal	house'and'window	bill'paint	sigdig.func	copyfile2.basic	Jack Baldridge
c64 comal 0.14	listerine	demo/muppet'keys	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	Michele Bostdorff
comalerrors	planet'earth	designs	~ data files ~	~for more info ~	Will Bow
ml.sizzle	program'outliner	guitar	~~~~~~~~~~~~~~~~	~ send sase to ~	Marc Clifford
hi	right'turn'only	helicopter	~ do not load ~	~~~~~~~~~~~~~~~~	Tony Granata
menu	simple'term	house	~~~~~~~~~~~~~~~~	~ comal users ~	Eric Haas
~~	sort'dir	linear'graph	comalhelp.txt	~ group, usa ~	Bill Howard
~comal programs~	square'clock	matching	programs.dat	~ 6041 monona ~	Jonnhy Laprarie
~~	terminal	moving'hi	window-rs232.obj	~ madison, wi ~	Len Lindsay
bridge'evaluator	uniform'poly	patterns	window-sprit.obj	~ 53716 ~	Kevin Page
c'curve	wumpus	rocket	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	Joe Postma
calendar	~~~~~~~~~~~~~~~~	shape'up	~basic program ~	~(608)222-4432 ~	David Powell
define'keys	~programs from ~	~~~~~~~~~~~~~~~~	~to copy files ~	~~~~~~~~~~~~~~~~	Kevin Quiggle
demo/window	~ toledo ~	~ functions ~	~~~~~~~~~~~~~~~~		Sid Seifelein
fancy'lettering	~ christian ~	~~~~~~~~~~~~~~~~	~ do not load ~		Robert Shingledecker
graphic'solution	~ high school ~	free.func	~ from comal ~		Stacy Sominski
guessmycode	~~~~~~~~~~~~~~~~	pi.func	~~~~~~~~~~~~~~~~		Lowell Toms
					David Warman
					Larry Winkles

contributors:
Richard Aurland
Phyrne Bacon
Richard Bain
M Bokhorst
Will Bow
Ray Carter
Marc Clifford
Lew Fleishman
Brian Grainger
Tony Granata
Eric Haas
Craig Hardy
Gerald Hobart
Bill Inhelder
James Kaminski
Dick Klingens
Len Lindsay
Kevin Quiggle
Robert Shingledecker
L Zabel

Today Disk #14 - Back

			95 Files	2 Blocks Free:
hi	program'outliner	db'report	- do not load -	----------------
----------------	puzzle	db'sort	----------------	- packages -
-comal programs-	right'turn'only	db'squash	dat.mansion	----------------
----------------	scene'magic	----------------	sng.polka/str	pkg.c128
1520-calendar	sort'dir	-data files for-	----------------	pkg.dualscreen
chip.modem	trig-art	- data base -	- functions -	pkg.pllprt
custom'dir	wumpus	----------------	----------------	pkg.test'package
demo/dualscreen	----------------	-copy with data-	func.get'input$	----------------
demo/spritemaker	- unprotected -	- base to new -	func.sigdig	-for more info -
guess'my'word	- version of -	- disk -	----------------	- send sase to -
guessmycode	-data base from-	----------------	- procedures -	----------------
inventory-1	- today disk 8 -	db'data	----------------	- comal users -
inventory-2	----------------	db'help.def	proc.steal'shape	- group, usa -
inventory-3	db'boot	db'help.lab	proc.swap	- 6041 monona -
listerine	db'define	db'help.rpt	----------------	- madison, wi -
multi'directory	db'help	db'name	- source code -	- 53716 -
old'mansion.pop	db'labels	----------------	----------------	----------------
package'maker	db'maintenance	- data files -	lib.labels	-(608)222-4432 -
petals	db'menu	----------------	src.test'package	----------------

contributors:

Richard Bain

John McCoy

Joel Rea

Robert Shingledecker

David Stidolph

Birrell Walsh

Modem Disk - September 86

			74 Files	4 Blocks Free:
boot c64 comal	----------------	chip.modem	-basic programs-	- ravics term -
c64 comal 0.14	-not completely-	func.get'input$	----------------	- features a -
comalerrors	- tested -	----------------	-load the first-	-buffer & clock-
ml.sizzle	----------------	- data file -	- file in each -	----------------
hi	proto'terminal	----------------	- section with -	ravics term8.4
----------------	----------------	dat.phone	- ',8' and run -	----------------
- comal 0.14 -	- comal 2.0 -	----------------	----------------	- xmodem -
----------------	----------------	- listed -	----------------	-file transfer -
simple'term	cterm	-procs and func-	- midwestterm -	- protocol and -
terminal	mini'term	----------------	- features bbs -	-supports 1650 -
vt-52.v4	terminal'2.0	func.pro'input$	----------------	----------------
----------------	unfinished'term	proc.fixmodem	midwestterm5.1	xmodem-auto
-prototype 0.14-	----------------	proc.modem	term.c1	xmodem/autodial
- program with -	-for superchip -	----------------	mlmid	xmodem/ml
-file transfer -	----------------	-public domain -	----------------	

Mark Finley provides this sophisticated Hazardous Materials system - HazMat for short.

Each truck that carries hazardous materials is required to post a four digit number inside a diamond box. The number identifies the material. This system could be used in a police station or fire department to quickly identify the material and provide handling warnings. Read the text files before trying to run the system.

ShareWare - Hazmat A

			77 Files	41 Blocks Free:
hi	ext.g17	ext.g44	ext.g68	txt.using'hazmat
----------------	ext.g18	ext.g45	ext.g69	----------------
- hazmat -	ext.g19	ext.g46	ext.g70	-for more info -
- programs -	ext.g20	ext.g47	ext.g71	- send sase to -
----------------	ext.g21	ext.g49	ext.g72	----------------
change'8to9	ext.g23	ext.g51	ext.g74	- group, usa -
hazmat	ext.g24	ext.g56	ext.g75	- comal users -
----------------	ext.g25	ext.g57	----------------	- 6041 monona -
- hazmat -	ext.g30	ext.g58	- fingercode -	- madison, wi -
- external -	ext.g33	ext.g61	----------------	- 53716 -
- procedures -	ext.g34	ext.g62	fingercode	----------------
----------------	ext.g36	ext.g63	----------------	-(608)222-4432 -
ext.g11	ext.g37	ext.g64	- text files -	----------------
ext.g13	ext.g38	ext.g65	----------------	
ext.g14	ext.g41	ext.g66	txt.doing'hazmat	
ext.g16	ext.g43	ext.g67	txt.fingercode	

This is the second part of the two disk HazMat system.

The system works on a dual drive with no disk swaps. On a single drive system disk swaps are minimized by keeping all the most used files on the same side of the disk. Since the system reports on the dangers the materials present, concerned citizens can use the program to find out just what materials are being transported through their city.

ShareWare - Hazmat B

			74 Files	9 Blocks Free:
ext.a	ext.h1	ext.p	ext.g12	ext.g55
ext.a1	ext.i	ext.p1	ext.g15	ext.g59
ext.b	ext.i1	ext.q	ext.g22	ext.g60
ext.b1	ext.j	ext.q1	ext.g26	----------------
ext.c	ext.j1	ext.r	ext.g28	-for more info -
ext.c1	ext.k	ext.r1	ext.g29	-sends sase to -
ext.d	ext.k1	ext.s	ext.g31	----------------
ext.d1	ext.l	ext.s1	ext.g32	- comal users -
ext.e	ext.l1	ext.t	ext.g35	- group, usa -
ext.e1	ext.m	ext.t1	ext.g39	- 6041 monona -
ext.f	ext.m1	ext.u	ext.g40	- madison, wi -
ext.f1	ext.n	ext.u1	ext.g42	----------------
ext.g	ext.n1	ext.w	ext.g48	-(608)222-4432 -
ext.g1	ext.o	ext.x	ext.g52	----------------
ext.h	ext.o1	ext.x1	ext.g53	

ShareWare Disk #1

```
                                                        70 Files          2 Blocks Free:
hi                faa10002          ~ knowledge ~      geology.qfmt      txt.sysop'notes
~~~~~~~~~~~~~~~~  mcm10001001       ~   bases   ~      geology.que       txt.using'coin
~ comal bbs ~     notes'to'sysop    ~~~~~~~~~~~~~~~~   geology.rec       ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~  olp10001          ~ audiodpt ~       geology.rfmt      ~for more info ~
~main programs ~  ~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~  ~ send sase to ~
~~~~~~~~~~~~~~~~  ~ proto-d ~       audiodpt.dat       ~ traffic calc ~  ~~~~~~~~~~~~~~~~
coinbbs           ~~~~~~~~~~~~~~~~  audiodpt.qfmt      ~~~~~~~~~~~~~~~~  ~ comal users ~
editor            proto-d'boot      audiodpt.que       traffic-calc      ~ group, usa ~
runbbs            proto-d'editor    audiodpt.rec       ~~~~~~~~~~~~~~~~  ~ 6041 monona ~
startup           proto-d'reader    audiodpt.rfmt      ~ text files ~    ~ madison, wi ~
wait'call         proto-d'writer    ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~  ~   53716   ~
~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~  ~ geology ~        txt.coin'message  ~~~~~~~~~~~~~~~~
~bbs data files~  ~ proto-d ~       ~~~~~~~~~~~~~~~~   txt.proto-d'how   ~(608)222-4432 ~
~~~~~~~~~~~~~~~~  ~  sample  ~      geology.dat        txt.proto-d'why   ~~~~~~~~~~~~~~~~
```

contributors:

Michael Erskine provides an expert system complete with small example "data bases"

Mark Finley provides traffic calc to assist with many of the calculations done at the scene of an accident

Richard Olivieri provides a COMAL Bulletin Board

Read and Run Disk #1

```
                                                        87 Files          4 Blocks Free:
hi                ml--story.5       ~ data files ~     west'asia.l       ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~  ml--story.6       ~~~~~~~~~~~~~~~~   west'europe.l     txt.madlibs
~  madlibs  ~     ml--story.7       africa.l           world.l           txt.maps
~~~~~~~~~~~~~~~~  ml--story.8       alaska.l           ~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~
~  programs  ~    ml--story.9       antarctica.l       ~sample output ~  ~for more info ~
~~~~~~~~~~~~~~~~  ml--story.10      australia'nz.l     ~~~~~~~~~~~~~~~~  ~ send sase to ~
madlibs           ml--words.1       britain.l          hrg.abtv'equi     ~~~~~~~~~~~~~~~~
ml--make words    ~~~~~~~~~~~~~~~~  canada.l           hrg.btv'equi      ~ comal users ~
ml--write story   ~   mapper   ~    china'korea.l      hrg.btv'ortho     ~ group, usa ~
~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~  east'europe.l      hrg.us'antarc     ~ 6041 monona ~
~  madlibs  ~     ~  program  ~     japan.l            hrg.usa'merc      ~ madison, wi ~
~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~  middle'america.l   mapdata.abtv'equ  ~   53716   ~
~ data files ~    mapper            postgroup.l        mapdata.btv'equi  ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~  scandanavia.l      mapdata.btv'orth  ~(608)222-4432 ~
ml--story.1       ~external proc ~  south'america.l    mapdata.us'antar  ~~~~~~~~~~~~~~~~
ml--story.2       ~~~~~~~~~~~~~~~~  south'asia.l       mapdata.usa'merc
ml--story.3       ext.accuprint     usa.l              ~~~~~~~~~~~~~~~~
ml--story.4       ~~~~~~~~~~~~~~~~  ussr.l             ~ text files ~
```

contributors:

Dennis Bouldin provides a very sophisticated map drawing system. Make sure to read the text files about it before trying to run the programs

David Warman provides Mad Lib - an exciting and funny story creation system. You may have heard about programs like this. Now you can try it out yourself.

User Group Disk #12

```
                                                        67 Files          1 Blocks Free:
boot c64 comal    cosine'surface    tp'look            tp'info.txt       ~for more info ~
comal 64 0.14     doodle            tp'make            ~~~~~~~~~~~~~~~~  ~ send sase to ~
comalerrors       ellipse           tp'menu            ~ data file ~     ~~~~~~~~~~~~~~~~
ml.sizzle         etch'a'sketch     tp'read'info       ~~~~~~~~~~~~~~~~  ~ comal users ~
hi                genetic'soup      tp'remove          files             ~ group, usa ~
menu              goalie            tp'roster          ~~~~~~~~~~~~~~~~  ~ 6041 monona ~
names.dat         probabity         tp'scratch         ~  function  ~    ~ madison, wi ~
~~~~~~~~~~~~~~~~  silly'game        tp'subject         ~~~~~~~~~~~~~~~~  ~   53716   ~
~comal programs~  surface           yatzee             fibonacci.func    ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~  tp'add            ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~  ~(608)222-4432 ~
bmc-bx80'dump     tp'boot           ~ text file ~      ~  procedure  ~   ~~~~~~~~~~~~~~~~
calendar'printer  tp'change         ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~
color'cone        tp'enter          ~ do not load ~    cardco'pad.proc
correlator        tp'initialize     ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~
```

contributors:

Bert Denaci

Michael Erskine

Eric Haas

Fraser Hunt

Tom Kuiper

Kevin Quiggle

Paul Ukleaja

User Group Disk #13

```
                                                        87 Files          4 Blocks Free:
hi                lecon'francais    goalie             plotter'pak       ext.converter-f
~~~~~~~~~~~~~~~~  lotto             silly'game         ~~~~~~~~~~~~~~~~  ext.converter-g
~comal programs~  math'drill        ~~~~~~~~~~~~~~~~   ~ data files ~    ext.converter-h
~~~~~~~~~~~~~~~~  password'maker    ~  requires  ~     ~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~
3'd-sampler       qubic'root        ~  printer  ~      ~ do not load ~   ~for more info ~
59th'st'bridge    scott'com         ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~  ~ send sase to ~
alpha'learn       show'characters   bioprint           dat.in.pr.s.1     ~~~~~~~~~~~~~~~~
ammonia           sman              class'forms        dat.in.pr.s.2     ~ comal users ~
autosoup          spcum             print'calendar     dat.lotto         ~ group, usa ~
bingo             spell             ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~  ~ 6041 monona ~
bridge            spr               ~  requires  ~     ~  external  ~    ~ madison, wi ~
clm               swatch'watch      ~   modem   ~      ~ procedures ~    ~   53716   ~
converter'menu    ~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~
cosine'surface    ~  requires  ~    cterm              ext.converter-a   ~(608)222-4432 ~
crazy'calvin      ~  joystick  ~    ~~~~~~~~~~~~~~~~   ext.converter-b   ~~~~~~~~~~~~~~~~
draw12            ~~~~~~~~~~~~~~~~  ~  requires  ~     ext.converter-c
graph1            ~control port 2~  ~ 1520 plotter ~   ext.converter-d
graphics'lines    ~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~   ext.converter-e
```

contributors:

Malcom Bridgham
Douglas Colpitts
A Conca
Bert Denaci
Peter Foiles
Eric Haas
Bob Hoerter
Frank Hunt
John Kleczewski
Abe Lebowitz
Don Pipkin
Kevin Quiggle
Nicholas Seachord
David Skinner
Scott Strool
Alan Tackett
Birrell Walsh

Order Form (MORE ITEMS ON OTHER SIDE)

Name:______________________________ SUBSCRIBER NUMBER:______ Sept 1986
(required for reduced prices-except new subs)
Street:______________________________ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:______________________________ VISA / MasterCard print card#/exp date:

VISA/MC #:______________________________exp date:______Signature:________________

Qnty Price List/Subscriber price-Item Description *(two disks may be supplied as one double sided disk)*
Prices subject to change *(all disks except IBM PC COMAL system are Commodore 1541 format)*

SUBSCRIPTIONS

[ ] _____ **COMAL Today newsletter-> How many? ____ Start with: 6 7 8 9 10 11 12 13 14 <-Circle one**
(each issue is 80 pages of articles, notes, tips, and program listings)
($18.95 for 6; $26.95 for 10 issues; >>> Canada add $1 per issue; >>> overseas add $5 per issue)
[ ] _____ $3.95/$1.50 COMAL Today backissue: circle #s wanted-> 5 6 7 8 9 -(ship add $1 each)
[ ] _____ $3.95/$2.95 COMAL Today backissue: circle #s wanted-> 10 11 12 13 14 -(ship add $1 each)
[ ] _____ $14.95/$12.95 COMAL Yesterday, first 4 issues of COMAL Today, spiral bound (ship add $3)
[ ] _____ **Today Disk subscription >>> How many disks (at least 6)? ____ Start with disk#_____**
(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
(Disk subscriptions must be 6 or more disks - single Today Disks are available - use above line)

SYSTEMS:

[ ] _____ $98.95/$94.95 Deluxe Cartridge Pak (with the black COMAL 2.0 cartridge)-(shipping add $5)
(also includes: Cartridge Tutorial Binder & disk and Cartridge Graphics & Sound book)
[ ] _____ $74.95/$69.95 Black COMAL 2.0 Cartridge, Plain (no books/no disk)-(shipping add $2)
[ ] _____ **$29.95/$24.95 Super Chip (for black COMAL 2.0 cartridge) with C128 support-(ship $1)**
[ ] _____ **$5/$5 Install and test Super Chip fee. Done for no charge if cartridge purchased at same time.**
[ ] _____ $19.95/$17.95 Programmers Paradise Pak (includes COMAL Today 5,6,7,8,9 & 10)(ship add $2)
(includes Fastloaded COMAL 0.14 system & COMAL From A To Z book)
[ ] _____ **$5/$5 option - only with Paradise Pak - (includes Tutorial disk, Best of Disk, Auto Run Demos)**
[ ] _____ **$29.95/27.95 COMAL 0.14 Starter Kit (5 disks, 2 books, 6 newsletters, more)-(ship add $4)**
(includes everything in Paradise Pak, plus $5 option, plus Graphics Primer disk/book)
[ ] _____ $350/$350 IBM Denmark PC COMAL 2.0 (Danish manual/COMAL Handbook)-(shipping add $5)
(system is in English, works on most IBM PC Compatibles with at least 256K)

DISKS:

[ ] _____ **$14.95/$9.75 Read & Run disk (COMAL 2.0 only)**
[ ] _____ **$14.95/$9.75 Shareware disk (for 2.0) -- Buy #1 -- Get #2 FREE. (Due July/Aug 86)**
[ ] _____ $14.95/$9.75 Best of COMAL 0.14 (new version - single side of disk)
[ ] _____ $14.95/$9.75 Auto RUN Demo and Tutorial Disk (perfect with COMAL Workbook)
[ ] _____ $14.95/$9.75 Bricks Tutorials (2 sided beginners disk - an expanded Tutorial disk)
[ ] _____ $14.95/$9.75 Utility Disk #1 for COMAL 0.14 (Utility Disk #2 see books section)
[ ] _____ $10/$9.75 User Group Disks 0.14 - Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 12
[ ] _____ $10/$9.75 User Group Disks 2.0 - Circle disks wanted: 11 13
[ ] _____ $29.95/$14.95 Set of all Cartridge Demo Disks (includes #1 #2 #3 & #4)
[ ] _____ $14.95/$9.75 Single Cartridge Demo Disk, Circle one wanted --> 1 2 3 4
[ ] _____ $14.95/$9.75 Slide Show Picture Disks - both #1 & #2 - (HRG type pictures & 0.14 system)
[ ] _____ $14.95/$9.75 Games Disk #1 (for 0.14 & 2.0)
[ ] _____ $14.95/$9.75 Typing Disk (for 2.0 only)
[ ] _____ $14.95/$9.75 Modem Disk (for 0.14 & 2.0) - programs on disk may change frequently
[ ] _____ $14.95/$9.75 Font Disk (for 0.14 & 2.0) - dozens of fonts (compatible with PaperClip too!)
[ ] _____ $94.05/$89.95 19 Disk Set (about 1000 programs for COMAL 0.14)-(shipping add $3)

=====
Total_____ + _____ Shipping (this side) (Canada & First Class add $1 extra per book)
Total_____ + _____ Shipping (other side)
===== + =====
Total_____ + _____ = US$_____ Total Paid (WI add 5% sales tax)-(shipping $2 minimum)

Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Order Form (MORE ITEMS ON OTHER SIDE)

Name:______________________________________ SUBSCRIBER NUMBER:_______ Sept 1986
(required for reduced prices-except new subs)
Street:_____________________________________ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:__________________________________ VISA / MasterCard print card#/exp date:

VISA/MC #:_______________________________exp date:______Signature:________________

Qnty Price List/Subscriber price-Item Description (all disks Commodore 1541 format) Prices subject to change

BOOKS:

(Canada & APO / FPO & First Class shipping add $1 more per book)

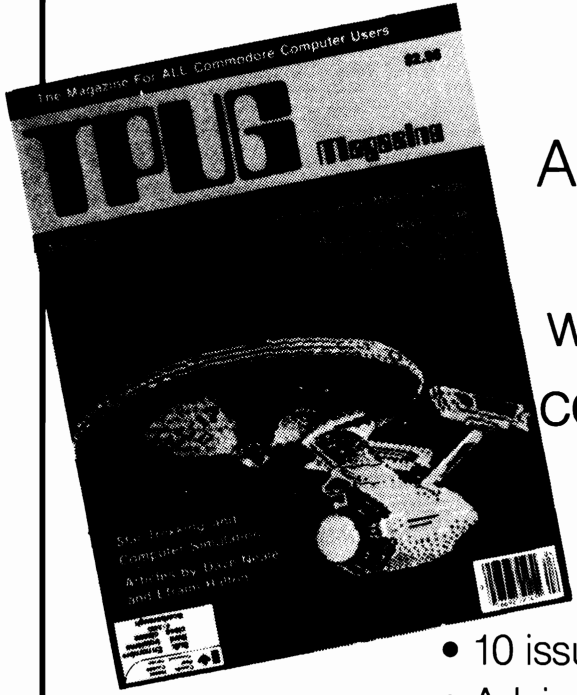
[ ] _____ $6.95/$4.95 COMAL Today--The INDEX, Kevin Quiggle (est 64 pages-due October 86)(ship $2)
[ ] _____ $14.95/$9.75/$7.95 The INDEX on disk! Let your computer find the article/author! (due Oct 86)
(COMAL Today issues 1-12, articles, notes, authors -- indexed! Buy book; add $7.95 for disk)
[ ] _____ $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)
[ ] _____ $17.95/$15.95 Starting With COMAL, Ingvar Gratte (212 pages)-(shipping add $3)
(Beginners text book, Jr/Sr High level, by Swedish professor)
[ ] _____ $19.95/$17.95 Foundations With COMAL, 2nd Edition, John Kelly (363 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Beginners text book, Jr/Sr High level, by Irish professor)
[ ] _____ $28.95/$26.95 Structured Programming With COMAL, Roy Atherton (266 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Intermediate text book, stressing modular programming, High School level, by British professor)
[ ] _____ $20.95/$18.95 Beginning COMAL, Borge Christensen (333 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Beginners text book, Elementary school level, by Danish professor, founder of COMAL)
[ ] _____ $17.95/$15.95 C64 Graphics With COMAL 0.14, Len Lindsay (170 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Reference book to COMAL 0.14 Graphics & Sprite commands, companion to COMAL Handbook)
[ ] _____ $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 COMAL Library Functions & Procedures, Kevin Quiggle (71 pgs, with disk)-(ship $2)
(For the 0.14 programmer, 140 procedures and functions ready to merge)
[ ] _____ $14.95/$12.95 Graphics Primer, Mindy Skelton (84 pages, with disk)-(shipping add $2)
(Beginners level tutorial to COMAL 0.14 graphics and sprites)
[ ] _____ $6.95/$4.95 Cartridge Graphics & Sound, Friends of Captain COMAL (64 pages)-(ship add $2)
(Mini reference book to all built in 2.0 cartridge package commands)
[ ] _____ $19.95/$17.95 COMAL 2.0 Packages, Jesse Knight (108 pages, with disk)-(shipping add $2)
(Advanced. For ML programmers- how to write your own packages- with C64sym & supermon)
[ ] _____ $19.95/$17.95 Packages Library, David Stidolph (76 pages, with disk)-(shipping add $2)
(Intermediate. 17 example packages, most with source code on disk- Smooth Scroll Editor too)
[ ] _____ $19.95/$17.95 Cartridge Tutorial Binder, Frank Bason, Leo Hojsholt (320 pgs, with disk)(ship $3)
(Beginners manual to everything in the COMAL 2.0 cartridge)
[ ] _____ $14.95/$12.95 COMAL Quick 0.14 & Utilities Set #2, Jesse Knight (24 pages, with disk)-(ship $2)
(Double sided disk for 0.14, one side FULL of printer utilities, including graphic screen dumps)

OTHER:

[ ] _____ OTHER: ____
[ ] _____ $3.95/$2.95 Keyboard Overlay for C64 COMAL 0.14 - Cheatsheet (shipping add $1)
=====
Total_____ + _____ Shipping = US$_____ Total Paid (WI add 5% sales tax)-(shipping $2 minimum)
USE OTHER SIDE FOR TOTAL if order includes items on that side
Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

TPUG

MORE THAN JUST ANOTHER COMPUTER MAGAZINE!



A membership in the world's largest computer club will provide you with:

- 10 issues of TPUG magazine
- Advice from experts like Jim Butterfield and Elizabeth Deal
- Access to our huge library of public domain software
- Access to our online help services
- All for only $25/yr.

TPUG
101 Duncan Mill
Suite G7
Don Mills, Ontario
CANADA
M3B 1Z3

JOIN US NOW!!

I want to join TPUG as an associate member.

Name ______	Home phone (____) ______
Address ______	Work phone (____) ______
City/Town ______	☐ Cheque enclosed
Prov/State ______	☐ Money order enclosed
Postal/Zip Code ______ Country ______	☐ Visa ☐ MasterCard
My computer equipment is: ______	Card# ______
______	Expiry ______
______	Signature ______

Expand Past Maximum Capacity!

The Tech/News Journal For Commodore Computers

At better book stores everywhere! Or 6 issues delivered to your door for just $15.00
That's 29% off the newstand price! (Overseas $21 U.S. Air Mail $40 U.S.)
The Transactor. 500 Steeles Ave. Milton, Ontario. L9T 3P7. 416 878-8438
Also check out The Transactor Disk; every program from each issue, in order as they appear
and The Complete Commodore Inner Space Anthology; over 2.5 million characters of reference information exclusively.
Included are memory maps for BASIC, COMAL, CBM disk drives, BBS numbers, machine language charts, Kernel routine summaries, printer commands, recording formats, I/O port maps, port pinouts, audio control tables, video reference charts, keyword values, commands for common software packages, MLM and assembler commands, *and there's more!*
To us, expansion knows no limits!