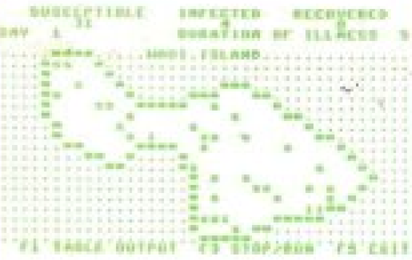
COMAL TODAY 19



```
  SUSCEPTIBLE    INFECTED    RECOVERED
      31             4           0
DAY  1                DURATION OF ILLNESS  5
.....................WOOD.ISLAND..........
 F1 TRACE OUTPUT  F3 STOP/RUN  F5 EXIT
```

Epidemic (page 67)

POWER DRIVER
page 12

CP/M GRAPHICS
page 10

SAMPLE BOOK PAGES
page 16

AFRICAN STONE GAME
page 64

ROTATING 3D IMAGE
page 53

3D SURFACE PLOTS
page 68

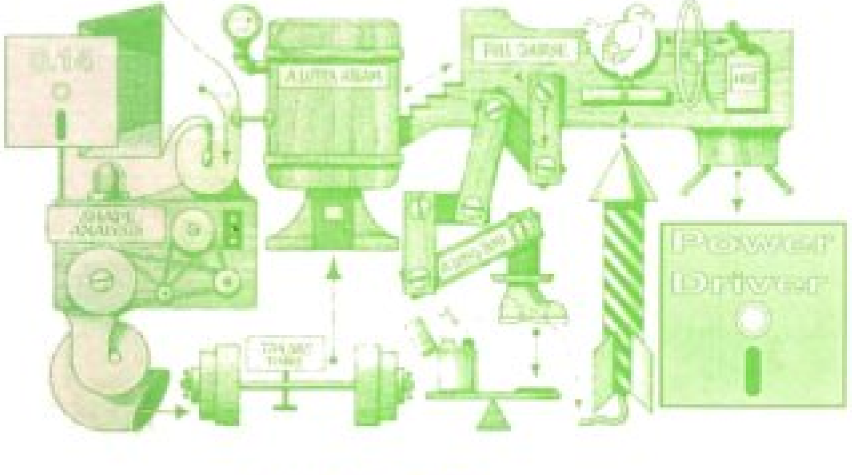
FILE RECOVERY
page 15

COMAL COLORING BOOK
page 45

SMARTER READER
page 69

1541 ALIGNER
page 69

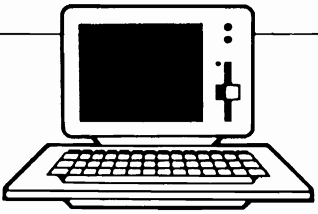
COMAL Today Staff prepares for SALE! See page 39



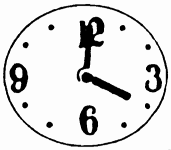
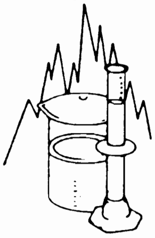
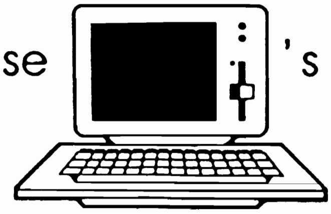
WHAT IS IT? (see page 12)

COMAL Today
6041 Monona Drive
Madison, WI 53716

If your label says
Last Issue: 19
You must renew now.
Use order form inside.

If you use a  , you need

The Computing Teacher

We publish articles from all over the in **The Computing Teacher** journal so that you'll get the best information. Information that's crystal clear, interesting ! and fun to use in the classroom. You can 1234 on us to have accurate, timely articles and to save you  and $ with our reviews and new product releases. We have columns for  , 1+3=4 , Logo, language arts and computers in the library. **The Computing Teacher**—for all those who use  's in the classroom.

ICCE, U of O, 1787 Agate Street, Eugene, OR 97403 USA

COMAL Today Issue 19 November 26, 1987

General

2 - Editor's Disk - Len Lindsay
3 - COMALites Unite
4 - Rumors
4 - Vote
4 - Doc Box
7 - Questions & Answers
8 - Letters
9 - COMAL Clinic - Christopher Laprise
11 - Bug Fixes
39 - Private Sale Insert
71 - Extra Programs
79 - Order Form

Beginners

5 - How To ...
43 - Hi Lo Game - Gary Franklin

Fun and Graphics

45 - COMAL Coloring Book - Dawn Hux
53 - Rotating 3D Image - Luther Hux
64 - African Stone Game - L W Zabel
67 - Epidemic - Bill Inhelder
68 - 3D Surface Plots - Sol Katz

2.0 Packages & Programming

10 - C128 CP/M Graphics - Richard Bain
12 - Power Driver - Captain COMAL
70 - Value

Applications

15 - File Recovery - Christopher Laprise
68 - Directory Boxes - Paul Keck
69 - Smarter Reader - Tim White
69 - 1541 Aligner - Normon Parron

Reference

6 - All Time Best Books
16 - Sample Book Pages
60 - UniComal for IBM PC - UniComal
72 - Power Driver Keywords
81 - Common COMAL Structures
82 - COMAL Benchmarks

Advertisers

ifc - Computing Teacher
63 - QLink
81 - Transactor

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

Artwork
Rhianon Lindsay

Contributors
Richard Bain
N Bakker
Doug Bittinger
Ray Carter
Borge Christensen
Captain COMAL
Herbert G Denaci
Gary Franklin
Jim Frogge
Eric Haas
Oren Hasson
Garrett Hughes
Dawn Hux
Luther Hux
Bill Inhelder
Alan Jones
Sol Katz
Paul Keck

Contributors
John Kelly
Jesse Knight
Christopher
Laprise
J William Leary
Ralph LeVine
Len Lindsay
Terry Mills
Kevin Quiggle
Richard Shagott
Todd Shagott
Gordon Shigley
David Stidolph
Frank Tymon
Melody Tymon
UniComal A/S
Tim White
L W Zabel

COMAL Today welcomes contributions of articles, manuscripts and programs which would be of interest to readers. All manuscripts and articles sent to COMAL Today will be treated as unconditionally assigned for publication and copyright purposes to COMAL Users Group, U.S.A., Limited and is subject to the Editor's unrestricted right to edit and to comment editorially. Programs developed and submitted by authors remain their property, with the exception that COMAL Users Group, U.S.A., Limited reserves the right to reprint the materials, based on that published in COMAL Today, in future publications. There will be no remuneration for any contributed manuscripts, articles or programs. These terms may be varied only upon the prior written agreement of the Editor and COMAL Users Group, U.S.A., Limited. Interested authors should contact the Editor for further information. All articles and programs should be sent to COMAL Users Group, U.S.A., Limited, 6041 Monona Drive, Madison, WI 53716. Authors of articles, manuscripts and programs warrant that all materials submitted are original materials with full ownership rights resident in said authors. No portion of this magazine may be reproduced in any form without written permission from the publisher. Local Users Groups may reprint material from this issue if credit is given to COMAL Today and the author. The opinions expressed in contributed articles are not necessarily those of COMAL Users Group, U.S.A., Limited. Although accuracy is a major objective, COMAL Users Group, U.S.A., Limited cannot assume liability for article or program errors.

Editor's Disk

C64 0.14 C64 2.0 IBM CP/M Power Driver

Finally! A runtime compiler for the C64. Our continuing support for disk loaded COMAL on the C64 hits a new high! **Power Driver** not only adds 21 new built-in commands, but includes a compiler for the system as well. **Power Driver** also is more literate. It accepts both UPPER and lower case for commands, and lists programs with the keywords in UPPER case. **Power Driver** is copyrighted and you may not give out copies, but you can get **your** copy free with our new **Power Box** collection of three disks of utilities and two disks of over 250 procedures and functions. See pages 12 & 72 plus the special sale offer on page 41.

National on-line COMAL support continues on QLink... now in a better location. To find us, sign on to QLink, go to the **CIN** (Commodore Information Network) then:

- choose **Commodore Community**
  - choose **Programmers Workshop**
    - choose **COMAL**

Our area includes a Question & Answer Message Base and download libraries for both 0.14 and 2.0. Plus we have our own conference room for our **monthly meetings** -- the **first Sunday** and **second Thursday** of each month:

QLink COMAL Meetings	
Sundays	Thursdays
November 1	November 12
December 6	December 10
--holiday--	January 14
February 7	February 11
March 6	March 10

We are having our fourth big sale! As our long time subscribers know, this means great deals... for a very limited time. You may be surprised at some of the deals! See page 39.

Our latest vote results show that most readers like hearing rumors. However, we don't think it should include the same rumor for three years!

I pulled out one of my older *almost final* Mytech IBM COMAL disks and noted the year on the label: 1984. Now we hear that the final is delayed until the end of the year... taking us into 1988. This also delays Mytech's Amiga and MacIntosh COMAL projects. The kicker is that Mytech COMAL has many **peculiarities**, and is not that fast. For example, after a RUN command, it takes 18 seconds to prepass our Data Base program (see *COMAL Today #15*, page 26) before it even starts to execute it. UniComal takes less than a second.

Meanwhile, we have negotiated a $200 discount from UniComal on their IBM PC COMAL 2.1. They want to sell to the professional market, so don't expect any further price reduction! Their COMAL is the fastest we have seen plus it is very well done. We programmed our order and inventory systems in UniComal IBM COMAL. If you like C64 COMAL 2.0 ... you will get the same look and feel with UniComal on the IBM, right down to the full screen editing!

Make sure you have a complete collection of *COMAL Today*. We are out of room, and can't store them much longer. It will be sad, but soon they will be recycled. Those old issues had some good information... and it still applies!

We found some! Real collectors items. The originals, not copies! We have a couple dozen copies of the very first issues of *COMAL Today* available now. See sale page 39.

New for CP/M COMAL: Richard Bain's book that shows how to make your own packages. similar to Jesse's C64 cartridge Package book.

NOTE: if you get this newsletter after November 1987, pages 39-42 may be missing... sorry, the sale is over. ■

COMALites Unite

This issue of *COMAL Today* proudly presents our __Fourth Private Sale__. We are offering more than just our best prices before the holidays... we have several new products as well.

Our feature item is the **Power Box**. This includes six disks of COMAL 0.14 procedures, functions, and utilities plus two books. In addition, we are including, absolutely free, the brand new **COMAL 0.14 Power Driver**. It adds commands to give disk based programmers many of the features in the COMAL cartridge. We have spent a great deal of time and effort to give the best possible support to COMAL programmers at a reasonable cost. We hope to get your support in return. Keep the COMAL tradition alive. See pages 12, 72, and sale page 41 for details.

Dick Hefner sent us a set of guitar tutor disks for COMAL 0.14. In the rush to put the newsletter together, we failed to fit in an article about the tutor, yet it should not be overlooked. It is a friendly system which plays songs while displaying the words and the chords on the screen. A beginning student can adjust the speed of the song to best fit his or her ability. See sale page 42.

Currently, we are putting the final touches on a **graphics** package for C128 CP/M COMAL. Our goal was to be as compatible as possible with the C64 cartridge. If you have waited to get CP/M COMAL because it lacked graphics, now is the time. See pages 10, 40 and 42 for details.

Two new books are at the press. Len Lindsay's __*Common COMAL Reference*__ (originally to be called COMAL Cross Reference) is an up to date reference for the versions of COMAL 2.0 available in North America. Richard Bain's __*CP/M COMAL Package Guide*__ tells Z80 programmers how to add their own packages to CP/M COMAL. See pages 18, 27, and sale page 41.

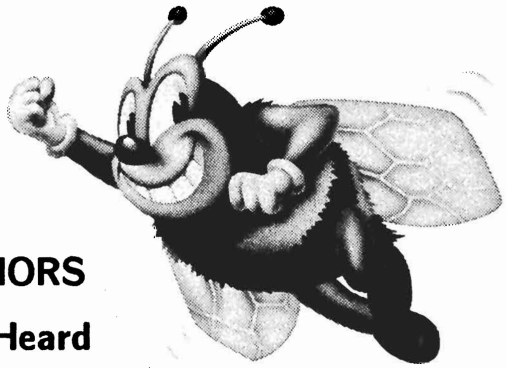
We regret to announce problems getting COMAL products from Europe, including the *Cartridge Tutorial Binder (COMAL 80)*, the C64 and C128 cartridges, and UniComal IBM PC COMAL. We should have our UniComal orders straightened out now. However, we were surprised to find that just the shipping and customs duties on the last set of *Tutorial Binders* we ordered came to about $18 per book! This doesn't even include the cost of the books themselves (not cheap) nor the overseas phone calls to order them, and check on the order several times. Please have patience with these orders. We do our best.

In talking to users we are surprised that some think we are making a lot of money on the COMAL 2.0 cartridges. This is understandable since the price is almost that of a new C64 and other cartridges are much less expensive. However, we make very little, if anything, on those cartridges. COMAL is a full 64K cartridge, not an 8K game cartridge.

Ordering COMAL 2.0 cartridges usually requires 3-4 overseas phone calls to Denmark (complete with "*please hold*") to order, arrange shipment, confirm, and check up on the delays. The cartridges are usually sent via two air carriers to a brokerage firm that arranges (and bills a lot for) the import paperwork. The broker then ships them to us.

All in all we have to pay three shippers, the brokerage firm, customs (the governments share) and the telephone company. On top of all that, the price we pay for the cartridge itself has been steadily increasing due to the decline of the value of the dollar (the dollar is worth about half of what it was worth when we introduced the cartridge... kind of like doubling our costs).

We think COMAL is worth all this trouble. We hope that you agree. ■



RUMORS

We Heard

by Captain "*Buzz*" COMAL

UniComal called to tell us that they are finalizing a COMAL for the new IBM PS/2 series computers. This includes support for the new VGA graphics. Meanwhile, they report that Germany is showing strong COMAL support.

Apple COMAL is being finalized. It is working on nearly all Apple II models, as well as the Laser 128 clone. The first release may even have turtle graphics.

Lego (the building blocks people, with headquarters in Denmark) are working on building block robots. There should be a controller system that runs under COMAL. The prototype that we saw was very impressive!

VOTES

Favorite articles in *COMAL Today #18*:

#1 - CP/M COMAL - page 14
#2 - Hamurabi - page 30
#3 - Rumors - page 5

What COMAL do you use?

0.14	■■■■■■■■■■■■■■■■■■■■■■■■
2.0	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
CP/M	■■■■■■■■■■■■■■■■■■
IBM	■■■■■■■■■■■■

Should we print rumors?

Yes	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
No	«none»

See page 39 for this issues VOTE questions.

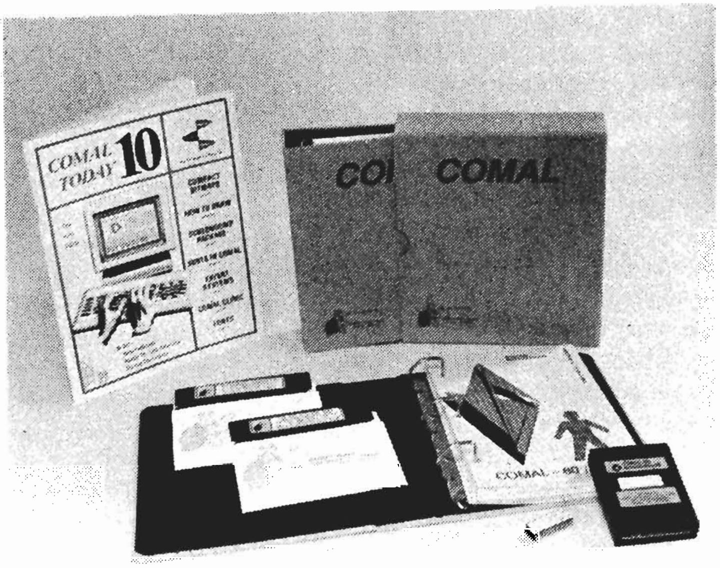
Doc Box

In *COMAL Today #17* we introduced the **Doc Box**. We spent months researching methods of "binding" our books, before adopting the **Doc Box** standard. Your favorable response indicates that we made a good decision.

For new subscribers... a **Doc Box** is a cloth bound 3 ring mini-binder for 5½ by 8½ pages, together with a matching **slip case**. This is the style IBM uses for their PC documentation. Each **Doc Box** can hold about 500 pages.

Many of our books are published as doc box style pages, punched to fit into the **Doc Box** (see the sample pages for full size reprints from the books). Since several books can fit into one **Doc Box**, the cover of each is designed as a tab sheet, making it easy to flip between the books.

The **Doc Box** met our criteria and seems to work well for our readers. It is an ideal size, pages stay open while reading, it is easy to expand and update, it is efficient, and the slip case keeps it's place on the shelf while you remove the binder. The photo below includes a **Doc Box**, in case you haven't seen one yet. ■



How To ...

Submit Articles and Programs

If you have any COMAL information, programs, or articles that you would like to share, send them 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 as a remark 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. Also include disk format: C64, IBM, CP/M, Apple, etc.

Articles should be submitted as standard text files on disk. If possible, also include a printout of each text file on the disk. Don't include any special formatting commands in your files (we have to delete them). We use special formatting for our LaserJet printer. (Currently we use Word Perfect 4.2 on our Zenith IBM compatible system).

Don't worry if you aren't a professional writer. We try to keep this newsletter informal. The information and programs are more important than perfect grammar. 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. ■

Type In Programs

Line numbers are required for your benefit in editing a program (but are irrelevant to a running program). Thus line numbers usually 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. When done:
   C64 COMAL 0.14: Hit «*return*» key twice
   C64 / C128 COMAL 2.0: Hit «*stop*» key
   CP/M COMAL 2.10: Hit «*esc*» key
   IBM PC COMAL: «*control*»+«*break*»
   Mytech: «*control*»+«C»

You may use both UPPER and lower case letters while entering a program. COMAL automatically makes keywords UPPER case and variable names lower case. (Note: C64 COMAL 0.14 may only use unshifted letters.) Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures, and COMAL will insert them for you. You **DO** have to type a space between keywords in the program.

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

Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz 0123456789'] [\ _

The «*left arrow*» key in the upper left corner of the C64/C128 keyboard is valid. COMAL 2.0 converts it into an underline. The C64/C128 computers use a «*British pound*» symbol in place of the \ backslash.

COMAL 0.14 users should read the notes on the next page. ■

more»

How To ... - - continued

COMAL 0.14 Special Instructions

C64 COMAL 0.14 was designed to be an introduction to COMAL. As such, it does not implement the full COMAL Kernal and is lacking some common enhancements. However, many COMAL 2.0 programs will also work in COMAL 0.14 with only slight modification. Here are some tips:

PAGE is not implemented. Add this procedure:

```
proc page
  print chr$(147),
endproc page
```

CURSOR, INPUT AT and **PRINT AT** are not implemented. Add this procedure:

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

Then for **INPUT AT** and **PRINT AT** first issue a **cursor** command, then use a normal **print**, **print using**, or **input** statement. Example:

```
  PRINT AT 9,1: USING "###":number; // 2.0
changes to:
  cursor(9,1)
  print using "###":num;
```

Additional procedures and functions that emulate COMAL 2.0 keywords are printed in past issues of *COMAL Today*. Over sixty procedures and functions are listed in *COMAL Today #15* starting on page 40. For example, if a COMAL 2.0 program includes VAL or STR, you can add a **val** function and **str** procedure as listed on page 49 in that issue. Also, remember not to use SHIFTed letters when typing in a program or filename. P.S. or consider using **Power Driver** without these worries. ■

All Time Best Books

This is it! The best selling COMAL books of all time (well, at least since 1984). We are very pleased that the number 1 book was written by the founder of COMAL, Borge Christensen.

Since we are reprinting a page from each book in this issue, we feel it is appropriate to report on the best sellers of all time. The results are interesting. If you are missing any of these books, the sale prices provide an ideal time to fill in the gaps on your COMAL Book Shelf.

1984 - 1987 Best Sellers

#1 - COMAL From A to Z
by Borge Christensen
64 page 0.14 reference book

#2 - COMAL Handbook
by Len Lindsay
479 page 2.0 / 0.14 reference book

#3 - Tutorial Binder
by Frank Bason & Leo Hojsholt
320 page 2.0 tutorial

#4 - COMAL Workbook
by Gordon Shigley
69 page 0.14 tutorial workbook

#5 - Introduction to COMAL 2.0
by J William Leary
272 page 2.0 textbook
with a 64 page answer book

#6 - Cartridge Graphics & Sound
by Captain COMAL's Friends
64 page 2.0 built-in package reference

#7 - COMAL 2.0 Packages
by Jesse Knight
108 page package tutorial/design guide

#8 - Beginning COMAL
by Borge Christensen
333 page textbook

#9 - COMAL Today - The INDEX
by Kevin Quiggle
52 page index to Comal Today 1-12

#10- Foundations with COMAL
by John Kelly
363 page textbook ■

Questions & Answers

Print Shop Converting

Question: Help. How do I convert the heart graphic from Print Shop to Print Master using the program in *COMAL Today #18*, page 46? I get a file not found error.

Answer from Terry Mills (program author): The graphics on the PS disk are not stored by their descriptive name (i.e., heart, rose, bells, etc.) but by number. If you look at the disk directory on your PS disk you will see files such as "i01", "i02", through "i60". These files correspond to the reference card that shows the birthday cake is graphic number 1, the heart is number 2, and so forth. Within Print Shop, when you tell it you want the heart, it knows to use the graphic found in the file "i02". I ran my converter program, used the original Print Shop disk as the source disk, and when prompted for a filename, typed in i02. It converted it on a data disk of my own, under the name i02.gra. I then loaded Print Master and called up the drawing pad (Print Master's graphic editor) and called in the file i02. (The program, when showing you the filenames, strips away the .gra extension.) I loaded it and it was in fact the heart shape from Print Shop.

I'd never come across this problem, nor thought much about the possibility of it occurring, as I save the graphics I like to a separate graphics disk from the graphics editor of each program before doing the conversions. For instance, from within Print Shop's editor I would load (get) the heart and then save it to my data disk under the name "heart". Similarly, I would save graphics from Print Master. Then I just converted the batch of them, having my favorite graphics in both formats on a single disk. But your thought to convert directly from the original disk is just as valid, if not more so. I probably should have included a warning to be aware of these filename peculiarities in my instructions for using the program. Hope this explains it satisfactorily.

New IBM PC COMAL 2.1

Question - Some of the IBM COMAL software which I have developed to help me in my work has generated interest among my co-workers. I believe some of them would be willing to get UniComal's IBM COMAL to be able to use my programs. However, before I can recommend UniComal to them, I'd like to know whether some of the deficiencies in the first release have been corrected. Are these flaws corrected?

1) Access to COM1: and COM2: by OPEN FILE or SELECT INPUT or SELECT OUTPUT fail.
2) TRACE is not implemented.
3) In the **graphics** package, **circle**, **arc**, and **plottext** are not implemented.
4) The **background** and **border** procedures in the **graphics** package give "Illegal color" for any value that I've tried.
5) Speed: how fast is UniComal IBM COMAL?

I hope you respond soon to these questions. I'd like to get some of my software distributed.

Answer: You will like the new UniComal 2.1 release. It has corrected all your concerns and right now, its price is reduced by $200. See the sale special on page 40. It corrects the flaws:

1) *COM1: and COM2: work. Even better, the new UniComal* **PLUS** *2.1 includes a modem communications package.*
2) *TRACE now works in the 2.1 release*
3) *The* **graphics** *package is improved. It works with CGA and EGA modes plus monochrome.* **Arc**, **circle**, *and* **plottext** *now work.*
4) **Background** *now works, using the color list included in the new UniComal manual.* **Border** *seems to have no effect on our amber monitor with our Zenith IBM PC compatible.*
5) *UniComal is very fast. It the fastest COMAL in North America and it is available for the IBM PC now. See the benchmark chart on the back cover.* ■

Letters

A Matter of Style

I am sure that you will get much hate mail over that one [*COMAL Today #18, page 17*]. Bill Inhelder is right on this issue. Maybe you just thought he was being defensive of his own code? **GOTO** should not be the issue. Clarity to people, and efficiency to machines are the only issues. Usually eliminating GOTO's will improve clarity. Structuring existing FORTRAN spaghetti code and simplifying the logic can make it run too slow to be used. I am not sure what **GOTO** or other structures do to COMAL execution speed. I think the clarity issue is more important here.

It is often better to structure the code in a manner that best fits the (il)logical thought process of the original analyst, rather than massaging the code into a more elegant form. Otherwise, when a change is needed both you and the analyst may have to start from scratch.

I spend a lot of time converting FORTRAN and BASIC programs to COMAL, and vice versa. I generally write a straight translation with GOTO's first, and make sure it runs right. Then I make simple structural changes like replacing FOR (DO) loops with **REPEAT** or **WHILE** loops. In FORTRAN a DO loop may be protected with an **IF** structure since many FORTRAN implementations will always execute the loop once regardless of the looping parameters. In this case I eliminate the **IF** structure and add a "//protected" to the **FOR** statement. When the COMAL code is translated back to FORTRAN the **IF** structure must be added back in. I keep integer variables even though this slows COMAL program execution. I try to avoid creating additional subroutines. I generally end up leaving in a few GOTO's so that there is enough correspondence between the original code and the COMAL code so that you can trace the program flow in both and make corresponding updates. It is easy to get carried away restructuring a program and make subtle mistakes or "improvements".

GOTO is simply an execution flow transfer structure that the programmer can use (sparingly) to improve clarity or meet program design requirements. A program with some GOTO's could very well be an excellent example of how to program rather than broken code that needs to be "fixed". Certainly enough code has been published in *COMAL Today* with "felony bad programming" even without GOTO's. I respectfully request that you drop your hard line anti **GOTO** policy. - Alan Jones, Ames, IA

We didn't get a lot hate mail on this issue, but we are willing to let the debate continue. The issue is primarily program clarity. Occasionally speed is a minor issue, but GOTO does not improve speed!

Maintaining the illogical thought process of the original programmer may be useful in team programming projects. However, in the context of a magazine or newsletter, it is rare that programs go through multiple revisions involving both the original author and the readers. It seems better to give the readers a cleaner program as a starting point. The original author can use the update, or his original as he sees fit.

For those who need to translate programs between computer languages, it is important to avoid features in one language which are not included in the other. I don't suspect FORTRAN programmers include a lot of recursion in their COMAL programs. But for a COMAL magazine, isn't it better to use the best COMAL code?

So for now, our policy is still the same. We take all the GOTO's out of the programs we release. - Richard Bain ■

COMAL Clinic

by Christopher Laprise

In *COMAL Today #16*, page 40, David Stidolph demonstrated how COMAL 2.0 turtle graphics are subject to roundoff error. This is apparently caused by the way the graphics package handles its X,Y coordinate system with floating point math. The reason why the package was designed this way probably relates to the **window** command. It defines the X and Y minimum and maximum values over any range (the default range is 320 by 200 for X and Y respectively). If the range for X and Y is changed (to 0-2 for example), then all of the systems pixels are still accessible through non-integer values.

A normal (default) screen:

```
199
!
!
0---------319
```

Thus, we have 320 pixels for the X axis and 200 pixels for the Y axis. But we may redefine our coordinate range like this:

```
2
!
!
0---------2
```

Now we must use real number 1.5 in order to plot a point 3/4 the way up the screen:

plot(0,1.5)

The system makes no attempt to round this off. A point is placed between 1 and 2 on the Y axis.

Despite all this, when using the default screen ranges, we can theoretically remedy the error factor in two easy ways. Both involve reading the current X,Y values with the built-in **Graphics** package variables **xcor** and **ycor**.

Run the modified version of David Stidolph's program first using the **PROC fix1**, and then changing that to **PROC fix2**, run it again.

```
USE graphics // initialize graphics
graphicscreen (0)
moveto(160,100) // move to center
POKE $c462,$5d // flip mode to draw
//
REPEAT
  FOR y:=1 TO 8 DO
    box
    right(45)
  ENDFOR y
UNTIL TRUE=FALSE // forever
// POKE $c4d2,$1d // disable flip mode
//
PROC box
  FOR x:=1 TO 4 DO
    right(90)
    forward(50)
    fix1 // change to fix2
  ENDFOR x
ENDPROC box
//
PROC fix1
  moveto(INT(xcor),INT(ycor))
ENDPROC fix1
//
PROC fix2
  a#:=xcor; b#:=ycor
  moveto(a#,b#)
ENDPROC fix2
```

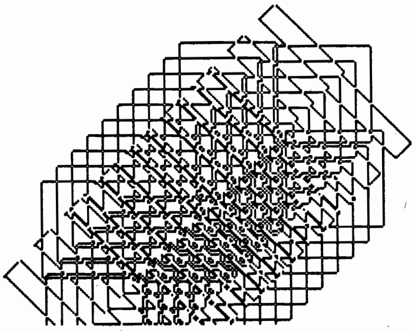
The first example (fix1) takes the **xcor** and **ycor** values and truncates their decimal *leftovers* or error. But what happens is that the error caused by the package routines can be either positive or negative. An error in the negative range will result in the coordinate being reevaluated one whole integer below the ideal integer goal. Even if the error causes the coordinate to be just a tenth below the ideal, the INT function will bring that figure down by another 0.9 below the ideal.

more»

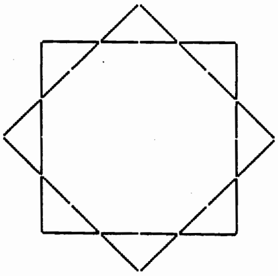
However, the second **PROC** uses a different method by using COMAL's integer variables. When a non-integer real number is fed into an integer variable, the value will be **rounded off** to the **nearest** integer, not **truncated** when that variable assumes a value.

So, when we have an ideal of 110 and we have a roundoff error of -0.15, the resulting value is 109.85. Using the integer method, 109.85 is truncated to 109. An error of 1 from the ideal of 110 is even worse than the original -0.15 error. But, when we assign the erroneous 109.85 to an integer variable such as a#, then a# becomes equal to 110, our ideal. [*Editor's note: this is a good reason to use integer variables.*]

Besides demonstrating a way to prevent error in repetitive turtle routines, this method also shows how COMAL can actually **estimate** numbers with the use of integer variables. This is not a well documented capability.



Screen dump using PROC fix1



Screen dump using PROC fix2 ■

CP/M Graphics

by Richard Bain

The C128 CP/M COMAL Graphics package started as a source code file given to us along with CP/M COMAL. However, the original source file lacked the code necessary to plot a point, thus none of the other commands worked. Ray Carter, an original tester of CP/M COMAL, wrote the code necessary to plot a point on the C128. I wrote the algorithms to make the graphics package more compatible with C64 COMAL 2.0. We currently have a working package, but are still adding minor enhancements. See sale page 42 to order this graphics package.

The graphics package gives 640*200 monocolor resolution. All the popular turtle graphics commands are included: plot, moveto, drawto, move, draw, right, left, forward, back, home, and fill. There are commands for circles, ellipses, drawing a line between two points, changing the screen coordinates, and more.

A few commands are still being worked on, plottext being the main one. This is complicated by the 16K video chip. It doesn't have enough memory to store a 16,000 byte graphics picture and the character set too. However, we can store the font in C128 memory while the graphics screen is active, so it can be restored later. We expect to use the stored character set for the plottext command. This should also yield a **Font** package for C128 CP/M COMAL.

The other main commands being worked on involve the drawing mode. Currently, the graphics commands **plot** points on the screen. We hope to also include drawing modes which erase or flip points.

Note, the C128 is the only CP/M system we have a graphics package working for. Those wanting graphics on other systems should contact the COMAL Users Group for information about adapting the source code. ■

Bug Fixes

Extended Print Using

Eric Haas has found a way to correct the bug in the print'using procedure in *COMAL Today #16* page 39. The procedure inserts commas every third place, regardless of your placement of commas in the format string.

The format string "####" is interpreted as ",###"; this does not allow enough room for a four digit number, even though four #'s were specified. The procedure calculates available space by taking the length of the format string and subtracting the number of commas. Since "####" has no commas, the procedure thinks it has room for four digits, but since it inserts a comma at the third place, there is really only room for three digits. This is the reason that print'using("####",1000) gives a substring error.

The following corrects the print'using procedure. Number the procedure by 10's starting with 10. Then type the following lines:

```
340 IF "." IN fmt$ THEN
350  c:=(("." IN fmt$)-1) DIV 4
360 ELSE
370  c:=lf DIV 4
375 ENDIF
```

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

COMAL-Flex

Ralph LeVine noticed that the *comal-flex(joy)* program on *Today Disk #18* seemed to load into the computer, but instead of being able to RUN the program, he lost control of the computer. The problem is that the program has both a FONT and a user defined package linked to it. COMAL has a bug loading this complicated combination. After linking the user defined package, COMAL closes the program file and tries to read the FONT definition from the keyboard. This problem is described in *COMAL Today #18*, page 12. Fortunately, the fix to this bug is also on *Today Disk #18*. As noted in the article on page 12 of last issue, we used the *re-linker* program to fix the *comal-flex(joy)* file. But, after making a last minute change to *comal-flex(joy)*, we forgot to fix the file again. Anyone wishing to use *comal-flex(joy)* will first need to run it through the *re-linker*. A corrected copy is on *Today Disk #19*.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Cute Cubes

The Cute Cubes program on page 52 of *COMAL Today #13* is by Oren Hasson. The newsletter gave him credit, but we put an incorrect name in the program listing on *Today Disk #13*.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Mandelbrot Revisited

In *COMAL Today #18*, page 55 we gave screen dumps for two mandelbrot drawings, but we gave the wrong information for the lower picture. To obtain that picture, use real center=-.1275, real range=0.05, imaginary center=-.89, autoscale=y. Also, both drawings require 50 iterations.

Ray Carter realized there was a faster way to calculate the data in Ted Groszkiewicz's program. To speed up the program, substitute the mandelbrot procedure with this one:

```
PROC mandelbrot
  az2:=0; bz2:=0
  WHILE count<it AND sizez<4 DO
    count:+1
    IF count>=it THEN dot
    ad:=az2-bz2; bd:=2*az*bz; az:=ad+ac
    bz:=bd+bc; az2:=az*az; bz2:=bz*bz
    sizez:=az2+bz2
    IF sizez>=4 THEN dot
  ENDWHILE
ENDPROC mandelbrot ■
```

COMAL 0.14 Power Driver

by Captain COMAL

COMAL Trivia #1: Since the introduction of COMAL 0.14 in November 1984, what has been the single largest request from users?
Answer: A COMAL Compiler

COMAL Trivia #2: What has been the second and third most requested things?
Answer: Added commands, especially GET$, VAL, and STR$; More memory.

Introducing the **COMAL 0.14 Power Driver**, disk loaded for the C64. Of course it loads and runs existing COMAL 0.14 programs. You expected that! The good news is the added commands, expanded user memory area and its **compiler**!

The **COMAL 0.14 Power Driver** has built in error messages and increased user program memory (over 15,000 bytes free). **COMAL 0.14 Power Driver** knows about UPPER and lower case letters. So now you can enter programs and commands in upper or lower case letters (it understands the command: **LisT**). This also makes it much easier to **ENTER** a COMAL 2.0 program from disk, since it is normally listed to disk using upper and lower case letters. **COMAL 0.14 Power Driver** even automatically converts all keywords in your programs to UPPER case, and all variable names to lower case, just like COMAL 2.0. For Example:

```
FOR delay:=1 TO amount DO NULL
```

The **COMAL 0.14 Power Driver** bridges the compatibility gap between COMAL 0.14 and COMAL 2.0, and let's you really get a feel for what the 2.0 cartridge is like. SAVEd COMAL 0.14 programs can be LOADed directly by the **COMAL 0.14 Power Driver**. There is no need to convert them or to first **LIST** them to disk. LISTed COMAL 2.0 programs can be ENTERed directly in the **COMAL 0.14 Power Driver** without worrying about upper case characters (although other changes may be required). The **COMAL 0.14 Power Driver** adds many commands to COMAL 0.14 which were previously only built into COMAL 2.0.

Power Driver Compiler

A purpose of a compiler is to turn a program into a stand-alone file that may be run independent of the programming language. That is what the **Power Driver runtime compiler** does. It creates a file containing your program and the COMAL code necessary to RUN it. While this will not speed up your program, it will give the program more memory to operate in (for larger arrays, etc). *Compiled* programs are loaded and run like a BASIC program (they no longer need COMAL or **Power Driver**).

New Commands

BITAND	bitwise AND
BITOR	bitwise OR
BITXOR	bitwise XOR
BYE	exit COMAL
CURCOL	return current cursor column
CURROW	return current cursor row
CURSOR	move cursor
DIR	disk directory in program
FREE	free memory available
GET$	get characters from a file
INKEY$	get from keyboard
INPUT AT	input from specific location
PAGE	clear screen / form feed
PI	3.14159266
PRINT AT	print at specific location
RANDOMIZE	randomize random numbers
SCAN	scan program for errors
SPC$	returns spaces
STR$	convert number into string
TIME	returns current jiffy count
VAL	convert string into number

These commands are not procedures you merge into your program - they are built in, and perform like their 2.0 counterpart. These commands were not chosen randomly; they are the most commonly used in 2.0 programs.

more»

Suppose you are writing a graphics program and need to plot numbers on the graphics screen. **PLOTTEXT** is the command you need to use, but it will only plot strings - not numbers. With **STR$** you can convert numbers to strings so they can be used with **PLOTTEXT**. **GET$** can be used to read bytes from a disk file and **TIME** can be used to time benchmarks or to provide proper pauses in programs. Since these commands are built in you no longer need to write procedures/functions to emulate them (saves valuable program memory). This makes your programs shorter and more readable.

With these added commands you may wonder if your COMAL 0.14 programs will still run under the **COMAL 0.14 Power Driver**. Yes they will. If you have a variable called PI in your program you can LOAD and RUN the program without problem. The variable **PI**, you see, is actually kept in your program as a token number - not as **PI**. The only time the variables name itself is important is when you enter a program line. This means that you could have variables or procedures named CURSOR, STR, VAL, etc. in your original program without problem.

[*We ran all 27 COMAL 0.14 programs from Today Disk #18 directly from the* **Power Driver**. *All worked except for the HI program (alters COMAL) and demo/load'font (places the font in the Power Driver program area). The HI program on Today Disk #19 works with both COMAL 0.14 and Power Driver.*]

Important Note:

The **COMAL 0.14 Power Driver** is not being sold. It is available free with each Power Box that we distribute to U.S.A. or Canada only. However, it is copyrighted. **You are not allowed to give away copies of the COMAL 0.14 Power Driver.** It cannot be included in Group libraries, BBS's or Networks. You can, however, compile your programs with the **Power Driver compiler** and distribute those programs royalty free.

Added Commands Summary

BITAND

flag:=num1 BITAND num2
This operator returns the bitwise ANDing of the integers num1 and num2. These integers must be in the range of plus/minus 2^31.

BITOR

flag:=num1 BITOR num2
This operator returns the bitwise ORing of the integers num1 and num2. These integers must be in the range of plus/minus 2^31.

BITXOR

flag:=num1 BITXOR num2
This operator returns the bitwise exclusive ORing of the integers num1 and num2. These integers must be in the range of plus/minus 2^31.

BYE

BYE
This command exits COMAL and returns to the built-in BASIC of the C64.

CURCOL

CURCOL
c:=CURCOL
This function returns the current screen column that the cursor is on (1-40).

more»

CURROW

CURROW
r:=CURROW
This function returns the current screen row that the cursor is on (1-25).

CURSOR

CURSOR «row»,«col»
CURSOR 1,1 // top left corner
CURSOR 0,5 // current row, 5th column
Places the cursor at the specified **row** and **column** on the text screen. A value of zero specifies that the current cursor row or column should be used.

DIR

DIR [«pattern$»]
DIR "=prg"*
Reads and prints the disk directory. An optional string may be included to specify drive 0: or 1: and/or wildcards. This command may be used from command mode or from a running program, but cannot be used to read the disk directory from unit 9. It is possible to pause the directory listing by pressing the «*space bar*».

Note: **CAT** works the same as before. It has an optional number for drive 0 or 1, but does not allow a string to specify wildcards.
CAT 0

FREE

memory:=FREE
This function returns the number of bytes available for user programs and data. It may be used within a running program to automatically size arrays within the limits of memory.

GET$

GET$(«filenumber»,«num'of'chars»)
WHILE NOT EOF(2) DO PRINT GET$(2,1),
This string function returns **num'of'chars** characters from file **filenumber**. The file must have been opened with the **OPEN** command before **GET$** can be used. If the end of the file is reached before all the characters are entered, then only those characters read will be returned and no error message is given.

INKEY$

INKEY$
WHILE INKEY$<>CHR$(13) DO NULL
This string function returns one character from the keyboard buffer. If no character is in the buffer, it blinks the cursor until a key is pressed and returns that character. This is useful with menus requiring a one letter choice.

INPUT AT

INPUT AT «row»,«col»:[«prompt»:] «input list»
INPUT AT 12,1: "Choice: ": c$
Inputs user data from the specified screen location. If zero is used for «row» or «col» then the current row or column is used. Note: this command does not allow the optional «max» input field length parameter available with COMAL 2.0. See also **CURSOR**.

PAGE

PAGE
Normally clears the text screen and places the cursor in the upper left corner of the screen. However, if output is to a printer (**SELECT OUTPUT** "**lp:**" has been issued, **PAGE** sends a form feed character: **CHR$(12)**.

PI

PI
*PRINT rad;"Radians =";rad*180/PI;"Degrees"*
This function returns 3.14159266.

PRINT AT

PRINT AT «row»,«col»: «print list»
PRINT AT 5,10: "Main Menu"
Prints the text and/or numbers at the specified row and column. If zero is used for «row» or «col» then the current row or column will be used. See also **CURSOR**.

more»

COMAL 0.14 Power Driver - continued

RANDOMIZE

RANDOMIZE [«seed»]
RANDOMIZE
Initializes the random number generator. The seed is optional. If it is included, the program will always generate the same random number sequence. If omitted, the current time (jiffy clock) is used for the seed.

SCAN

SCAN
Checks for structure errors in a program, without executing the program. It also adds identifier names to **ENDFOR**, **ENDPROC**, and **ENDFUNC** if they were omitted. After a successful SCAN you may call procedures and functions in the immediate mode.

SPC$

SPC$(«length»)
PRINT "[",SPC$(15),"]"
Returns a string containing the number of spaces specified by «length».

STR$

STR$(«num»)
p$=STR$(PI)
This string function returns the same characters that represent the number.

TIME

TIME [«num»]
TIME 0
PRINT TIME
A command and a function. As a command it sets the built in jiffy clock. As a function it returns the current value of the jiffy clock. The jiffy clock counts in 1/60ths of a second.

VAL

VAL(«text$»)
value:=VAL(p$)
This function returns the number that the string text$ represents. Note: if text$ is an invalid number, or has illegal characters, a zero is returned. ■

File Recovery

by Christopher Laprise

This program was written to recover some files after I had NEWed the directory on a program disk. (This program will NOT help if the disk has been formatted.) It's unlike most unscratch programs in that it does not use the directory to find the deleted programs. Instead, it scans the entire disk for blocks which look like the beginnings of a certain type of file. It reconstructs the file, block-by-block until it reaches the last block of the file. After that, you need to swap disks so the program can re-write the file to disk. **Do not have the program write the file to the same disk!** Doing that would write over files you are recovering.

After each file is read in, you will be asked for a filename. You could name them in sequence like: *a,b,c,d* .... Once the recovery procedure is complete and you find out what each file is, you can **RENAME** them. Once you've gotten all the different file types off the disk that you can (with subsequent RUNs), you can copy the files you recovered back to the original disk.

The program is preset to search for COMAL 2.0 saved (*prg*) files. However, you can determine what kind of files the program will seek out by changing the contents of the search string in the program. For instance, the program can tell whether a certain block is the beginning of a COMAL 2.0 program because all COMAL 2.0 saved program files begin with the ASCII values: 255, 255, 2. To search for a BASIC 2.0 file you can change the value of reckon$ to 1, 8. To find most listed COMAL programs, use "0010". Be careful that the **OPEN** statement in procedure store (which saves the files) is set to write the recovered file to the proper file type for the files you're recovering (*prg*, *seq*, *usr*).

Further reference:

Unscratching Files, COMAL Today #13, page 31
Fix Disk Errors, COMAL Today #11, page 16 ■

Sample Book Pages

The order form in each issue of *COMAL Today* includes a long list of books. We believe many COMAL users may be intimidated by this list. It is hard to decide if you want or need a book based on a one line description. It can be helpful to talk to friends or user group members for suggestions about which books are best. We can offer advice if you call us. However, the best way to choose a book often involves a trip to the book store to browse through books you are interested in. Since this is not possible for most COMAL books, we decided to bring the book store to you, so to speak. The following pages are reprints of one page from each of the COMAL books.

We offer books in several categories. There are general COMAL books which apply to all versions of COMAL. These include our reference books such as the *COMAL Cross Reference* and textbooks such as *Foundations in Computer Studies*. Younger COMAL programmers may prefer *Beginning COMAL*, while programmers with a high school math background may find *Introduction to Computer Programming With COMAL 2.0* more appropriate.

There are books which stress the special features of one version of COMAL. The *COMAL From A to Z* book is specific to COMAL 0.14 and the *Cartridge Graphics and Sound* book is specific to the C64 COMAL 2.0 cartridge. In general, it is best to have a book specific to the version of COMAL you are using, plus at least one general COMAL book.

In addition to the types of books mentioned above, there are many specialty books. Most, but not all of these books apply primarily to C64 COMAL 2.0, but as COMAL is a standard language, users of other versions of COMAL can often make use of some of the material.

The *COMAL Workbook* takes the beginning COMAL 0.14 user from loading COMAL into the computer for the first time to writing an inventory program. *Captain COMAL Gets Organized* teaches modular programming concepts while creating a COMAL 0.14 disk organization system. *The Library of Functions and Procedures* and the *Utilities Book 2* can provide many useful routines or building blocks for your programs.

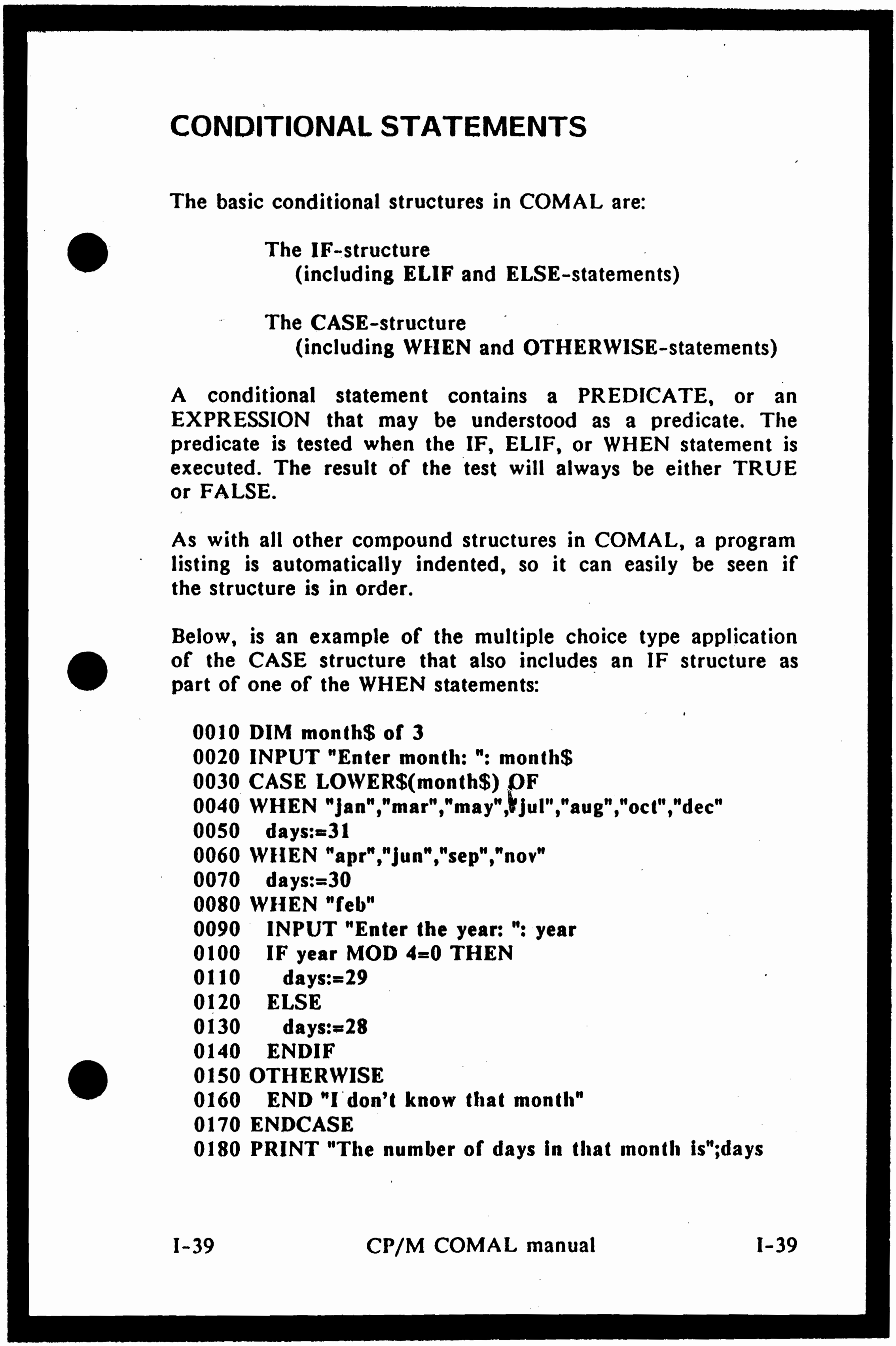
C64 COMAL 2.0 users can run *Graph Paper*, a high school level algebra and function plotting system. *Three Programs in Detail* shows the steps the author followed to create a black book address program, an accounting program, and a BBS program. The *COMAL Collage* demonstrates how to use C64 features such as joysticks, light pens, sound (including a Morse code), graphics, and sprites.

There are three books to help with C64 COMAL 2.0 packages. *Packages Library Vol 1* and *Packages Library Vol 2* are primarily intended to help COMAL programmers use the packages that other people have written. Dozens of packages are fully documented so that programmers can include the packages in their own programs without having to know machine code. Those who do use machine code will also find the source code included for most packages very valuable. *COMAL 2.0 Packages* is the standard reference guide for those who want to write their own machine code for COMAL.

CP/M COMAL is too new to have many books written for it specifically. The CP/M manual along with a general COMAL text book should be enough to keep people busy for now. The first book specific to CP/M COMAL is the *CP/M Package Development Guide*. It details the format required for a package complete with an example and step by step instructions.

COMAL Today - the Index is essential for anyone trying to find anything in our back issues (1-12) of *COMAL Today*. Nearly 5000 cross-indexed entries quickly point the way to those wonderful, but forgotten, articles.

more»



CONDITIONAL STATEMENTS

The basic conditional structures in COMAL are:

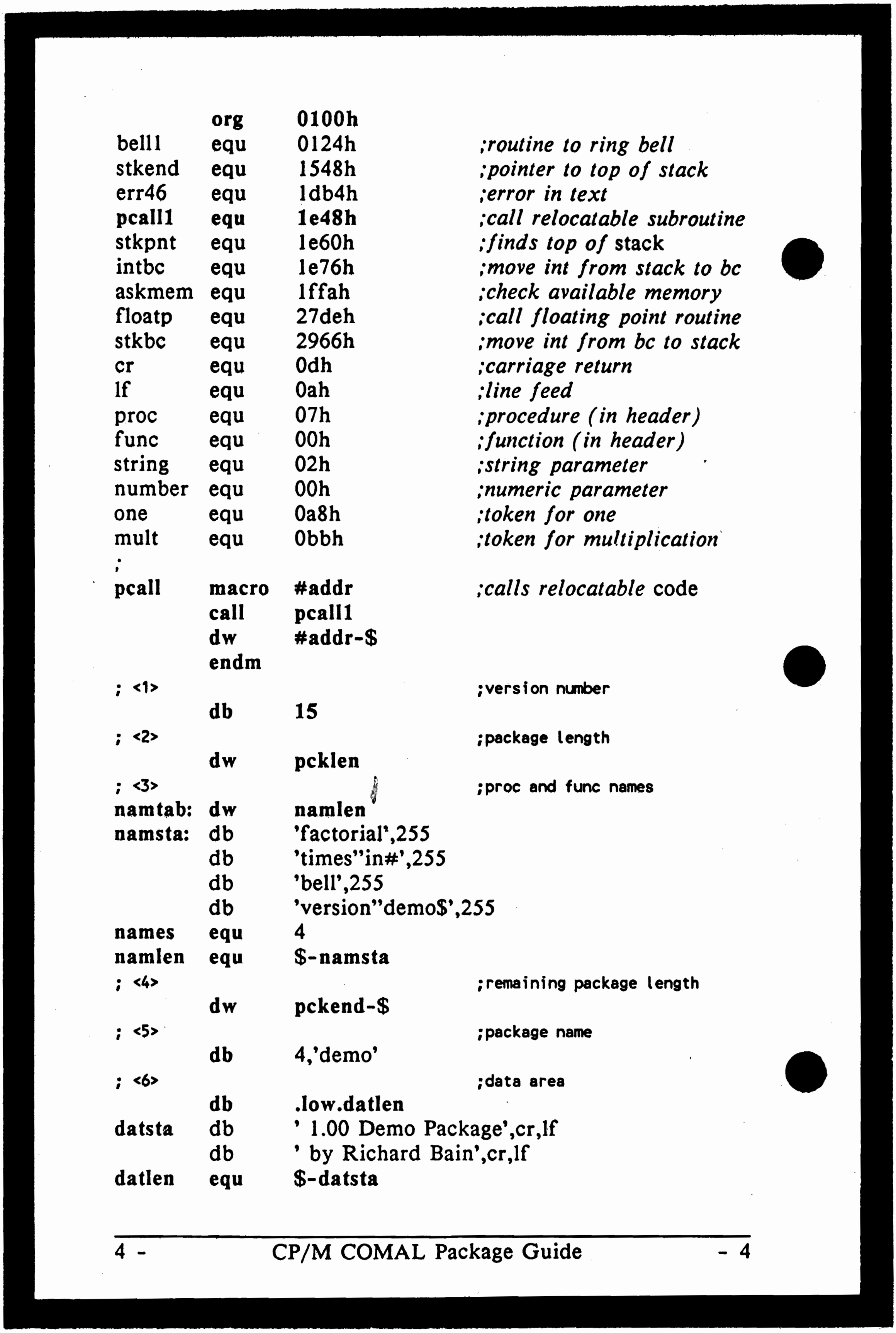
> The IF-structure
> (including ELIF and ELSE-statements)
>
> The CASE-structure
> (including WHEN and OTHERWISE-statements)

A conditional statement contains a PREDICATE, or an EXPRESSION that may be understood as a predicate. The predicate is tested when the IF, ELIF, or WHEN statement is executed. The result of the test will always be either TRUE or FALSE.

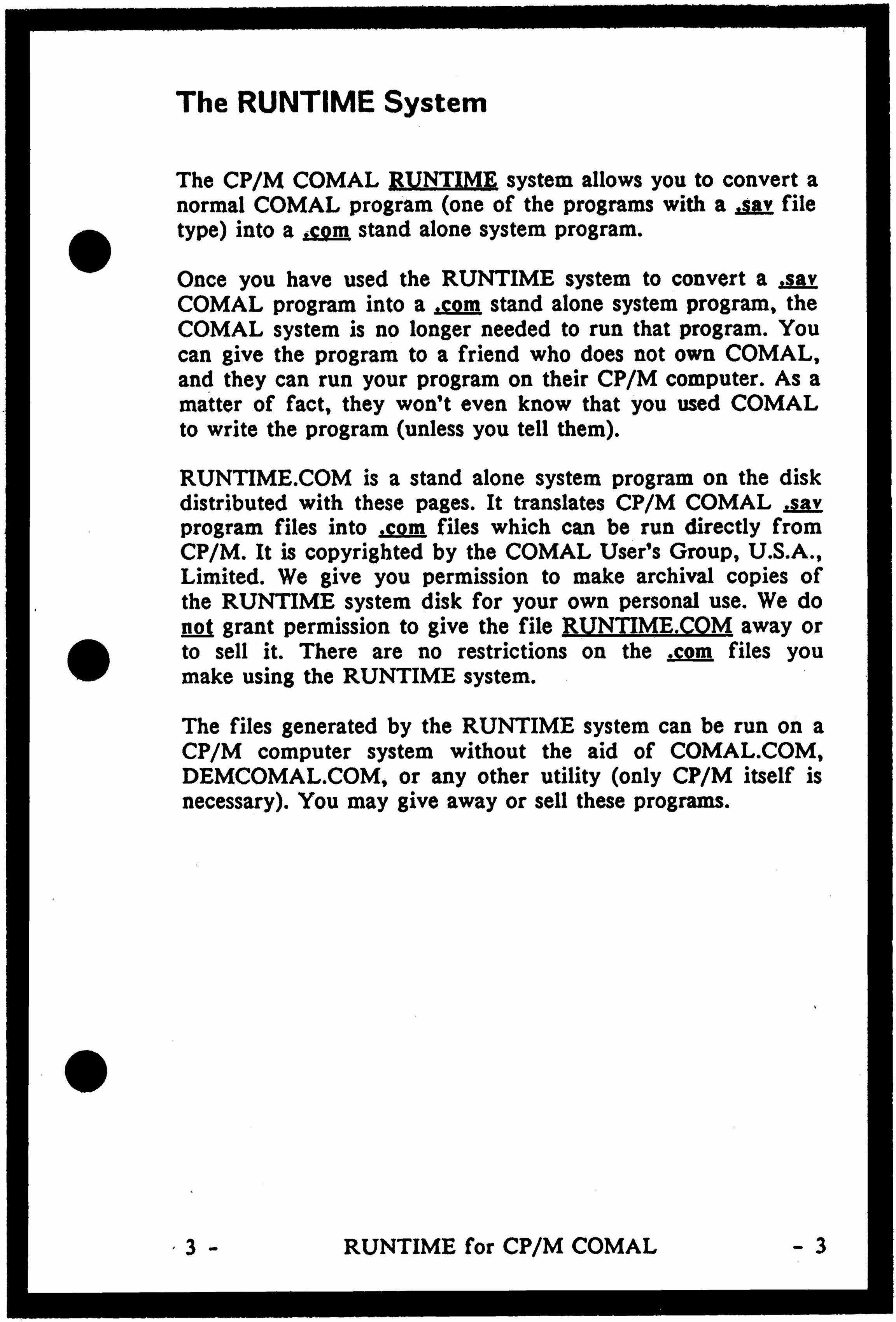
As with all other compound structures in COMAL, a program listing is automatically indented, so it can easily be seen if the structure is in order.

Below, is an example of the multiple choice type application of the CASE structure that also includes an IF structure as part of one of the WHEN statements:

```
0010 DIM month$ of 3
0020 INPUT "Enter month: ": month$
0030 CASE LOWER$(month$) OF
0040 WHEN "jan","mar","may","jul","aug","oct","dec"
0050   days:=31
0060 WHEN "apr","jun","sep","nov"
0070   days:=30
0080 WHEN "feb"
0090   INPUT "Enter the year: ": year
0100   IF year MOD 4=0 THEN
0110     days:=29
0120   ELSE
0130     days:=28
0140   ENDIF
0150 OTHERWISE
0160   END "I don't know that month"
0170 ENDCASE
0180 PRINT "The number of days in that month is";days
```



```
          org     0100h
bell1     equ     0124h             ;routine to ring bell
stkend    equ     1548h             ;pointer to top of stack
err46     equ     1db4h             ;error in text
pcall1    equ     1e48h             ;call relocatable subroutine
stkpnt    equ     1e60h             ;finds top of stack
intbc     equ     1e76h             ;move int from stack to bc
askmem    equ     1ffah             ;check available memory
floatp    equ     27deh             ;call floating point routine
stkbc     equ     2966h             ;move int from bc to stack
cr        equ     0dh               ;carriage return
lf        equ     0ah               ;line feed
proc      equ     07h               ;procedure (in header)
func      equ     00h               ;function (in header)
string    equ     02h               ;string parameter
number    equ     00h               ;numeric parameter
one       equ     0a8h              ;token for one
mult      equ     0bbh              ;token for multiplication
;
pcall     macro   #addr             ;calls relocatable code
          call    pcall1
          dw      #addr-$
          endm
; <1>                               ;version number
          db      15
; <2>                               ;package length
          dw      pcklen
; <3>                               ;proc and func names
namtab:   dw      namlen
namsta:   db      ’factorial’,255
          db      ’times”in#’,255
          db      ’bell’,255
          db      ’version”demo$’,255
names     equ     4
namlen    equ     $-namsta
; <4>                               ;remaining package length
          dw      pckend-$
; <5>                               ;package name
          db      4,’demo’
; <6>                               ;data area
          db      .low.datlen
datsta    db      ’ 1.00 Demo Package’,cr,lf
          db      ’ by Richard Bain’,cr,lf
datlen    equ     $-datsta
```



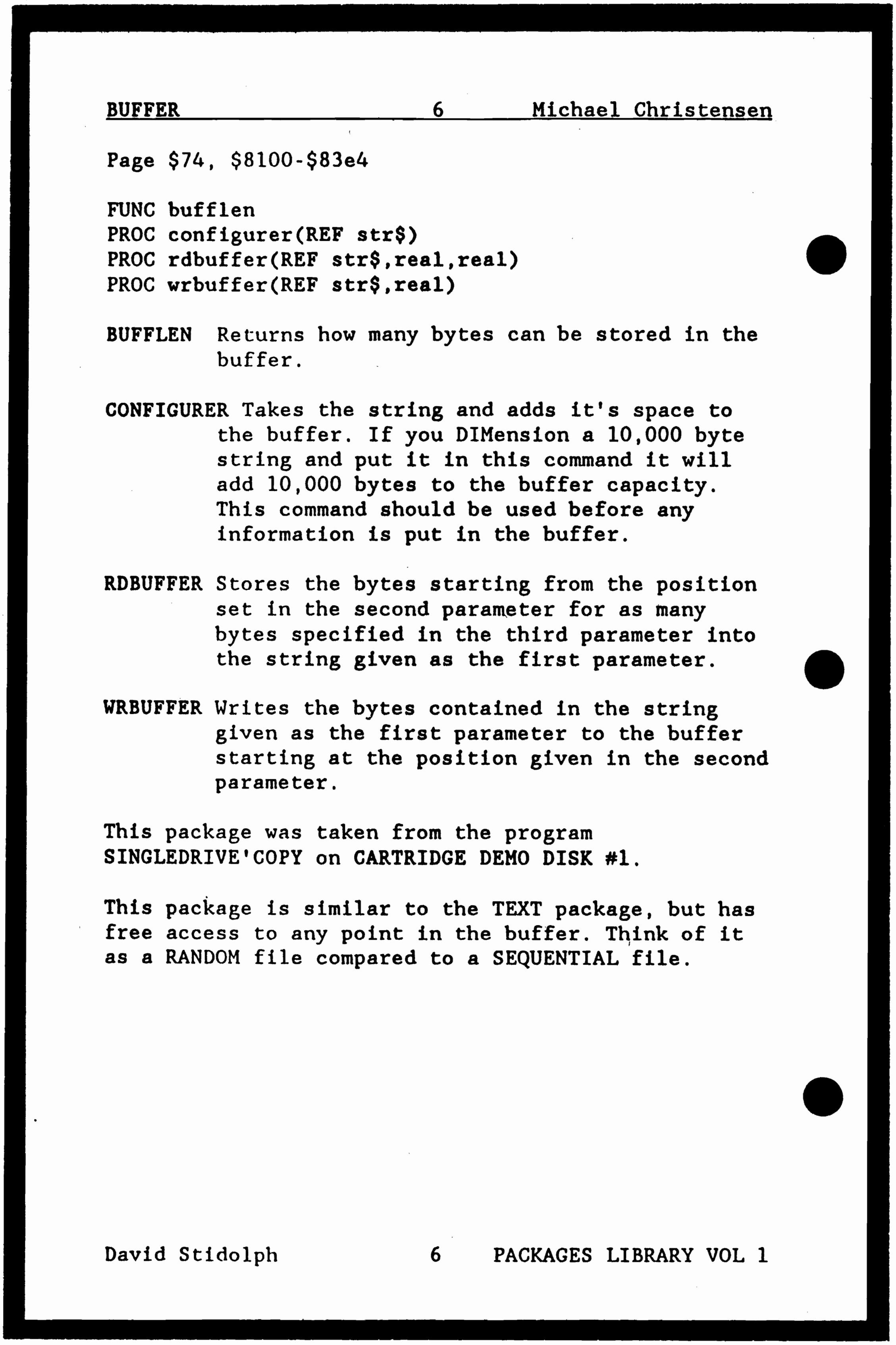
The RUNTIME System

The CP/M COMAL RUNTIME system allows you to convert a normal COMAL program (one of the programs with a .sav file type) into a .com stand alone system program.

Once you have used the RUNTIME system to convert a .sav COMAL program into a .com stand alone system program, the COMAL system is no longer needed to run that program. You can give the program to a friend who does not own COMAL, and they can run your program on their CP/M computer. As a matter of fact, they won't even know that you used COMAL to write the program (unless you tell them).

RUNTIME.COM is a stand alone system program on the disk distributed with these pages. It translates CP/M COMAL .sav program files into .com files which can be run directly from CP/M. It is copyrighted by the COMAL User's Group, U.S.A., Limited. We give you permission to make archival copies of the RUNTIME system disk for your own personal use. We do not grant permission to give the file RUNTIME.COM away or to sell it. There are no restrictions on the .com files you make using the RUNTIME system.

The files generated by the RUNTIME system can be run on a CP/M computer system without the aid of COMAL.COM, DEMCOMAL.COM, or any other utility (only CP/M itself is necessary). You may give away or sell these programs.



Page $74, $8100-$83e4

```
FUNC bufflen
PROC configurer(REF str$)
PROC rdbuffer(REF str$,real,real)
PROC wrbuffer(REF str$,real)
```

BUFFLEN Returns how many bytes can be stored in the buffer.

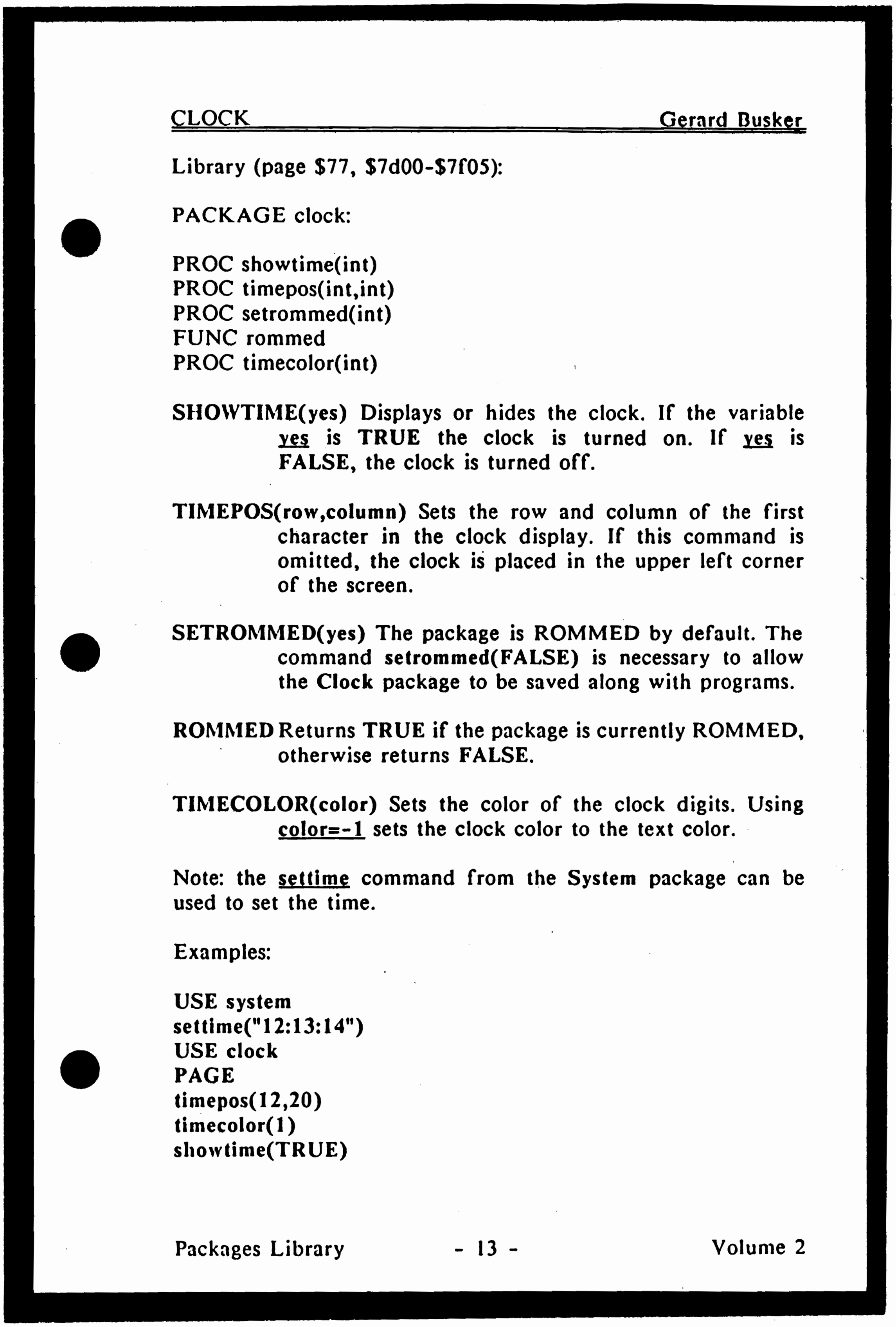
CONFIGURER Takes the string and adds it's space to the buffer. If you DIMension a 10,000 byte string and put it in this command it will add 10,000 bytes to the buffer capacity. This command should be used before any information is put in the buffer.

RDBUFFER Stores the bytes starting from the position set in the second parameter for as many bytes specified in the third parameter into the string given as the first parameter.

WRBUFFER Writes the bytes contained in the string given as the first parameter to the buffer starting at the position given in the second parameter.

This package was taken from the program SINGLEDRIVE'COPY on CARTRIDGE DEMO DISK #1.

This package is similar to the TEXT package, but has free access to any point in the buffer. Think of it as a RANDOM file compared to a SEQUENTIAL file.



Library (page $77, $7d00-$7f05):

PACKAGE clock:

```
PROC showtime(int)
PROC timepos(int,int)
PROC setrommed(int)
FUNC rommed
PROC timecolor(int)
```

SHOWTIME(yes) Displays or hides the clock. If the variable yes is TRUE the clock is turned on. If yes is FALSE, the clock is turned off.

TIMEPOS(row,column) Sets the row and column of the first character in the clock display. If this command is omitted, the clock is placed in the upper left corner of the screen.

SETROMMED(yes) The package is ROMMED by default. The command setrommed(FALSE) is necessary to allow the Clock package to be saved along with programs.

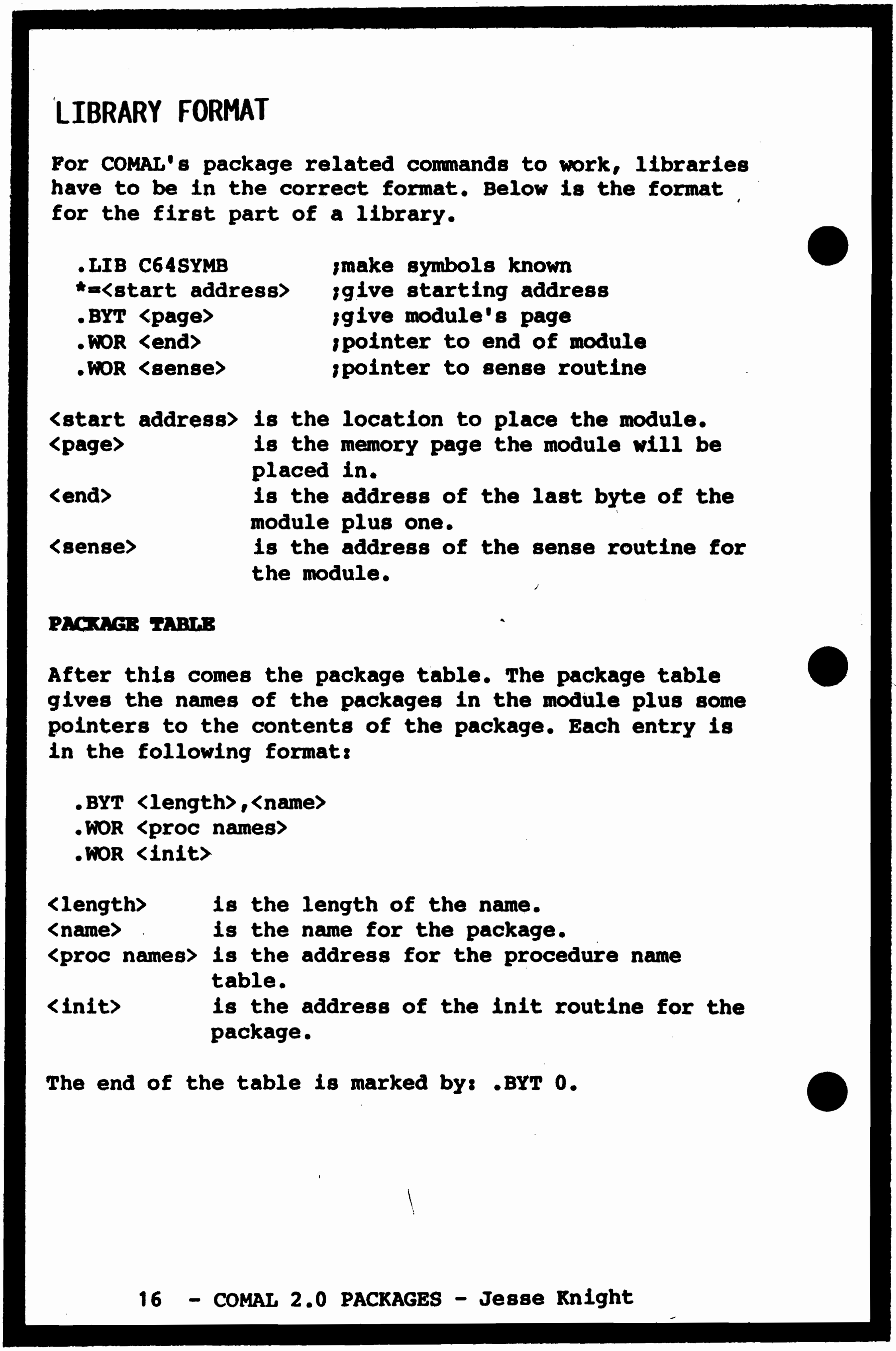
ROMMED Returns TRUE if the package is currently ROMMED, otherwise returns FALSE.

TIMECOLOR(color) Sets the color of the clock digits. Using color=-1 sets the clock color to the text color.

Note: the settime command from the System package can be used to set the time.

Examples:

```
USE system
settime("12:13:14")
USE clock
PAGE
timepos(12,20)
timecolor(1)
showtime(TRUE)
```



LIBRARY FORMAT

For COMAL's package related commands to work, libraries have to be in the correct format. Below is the format for the first part of a library.

```
  .LIB C64SYMB          ;make symbols known
  *=<start address>     ;give starting address
  .BYT <page>           ;give module's page
  .WOR <end>            ;pointer to end of module
  .WOR <sense>          ;pointer to sense routine
```

<start address> is the location to place the module.
<page> is the memory page the module will be placed in.
<end> is the address of the last byte of the module plus one.
<sense> is the address of the sense routine for the module.

PACKAGE TABLE

After this comes the package table. The package table gives the names of the packages in the module plus some pointers to the contents of the package. Each entry is in the following format:

```
  .BYT <length>,<name>
  .WOR <proc names>
  .WOR <init>
```

<length> is the length of the name.
<name> is the name for the package.
<proc names> is the address for the procedure name table.
<init> is the address of the init routine for the package.

The end of the table is marked by: .BYT 0.

To obtain a given number of grid rectangles counting vertically, divide the number of grids you desire into 120. The result is the number of pixels for the vertical dimension of each grid. Enter this number for V as described above.

For example to obtain graph paper with 10 grids horizontally and 10 vertically, divide 200 by 10 giving H=20 pixels, then divide 120 by 10 giving V=12 pixels.

Figure 15 shows a grid design with a horizontal grid dimension of 20 pixels and a vertical grid dimension of 12 pixels...well, almost!

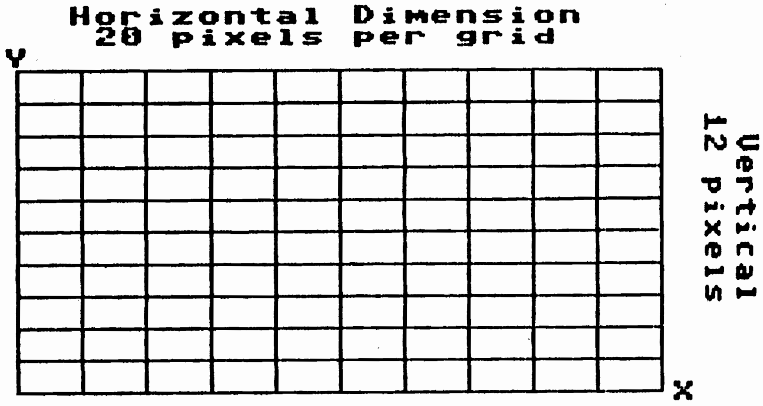
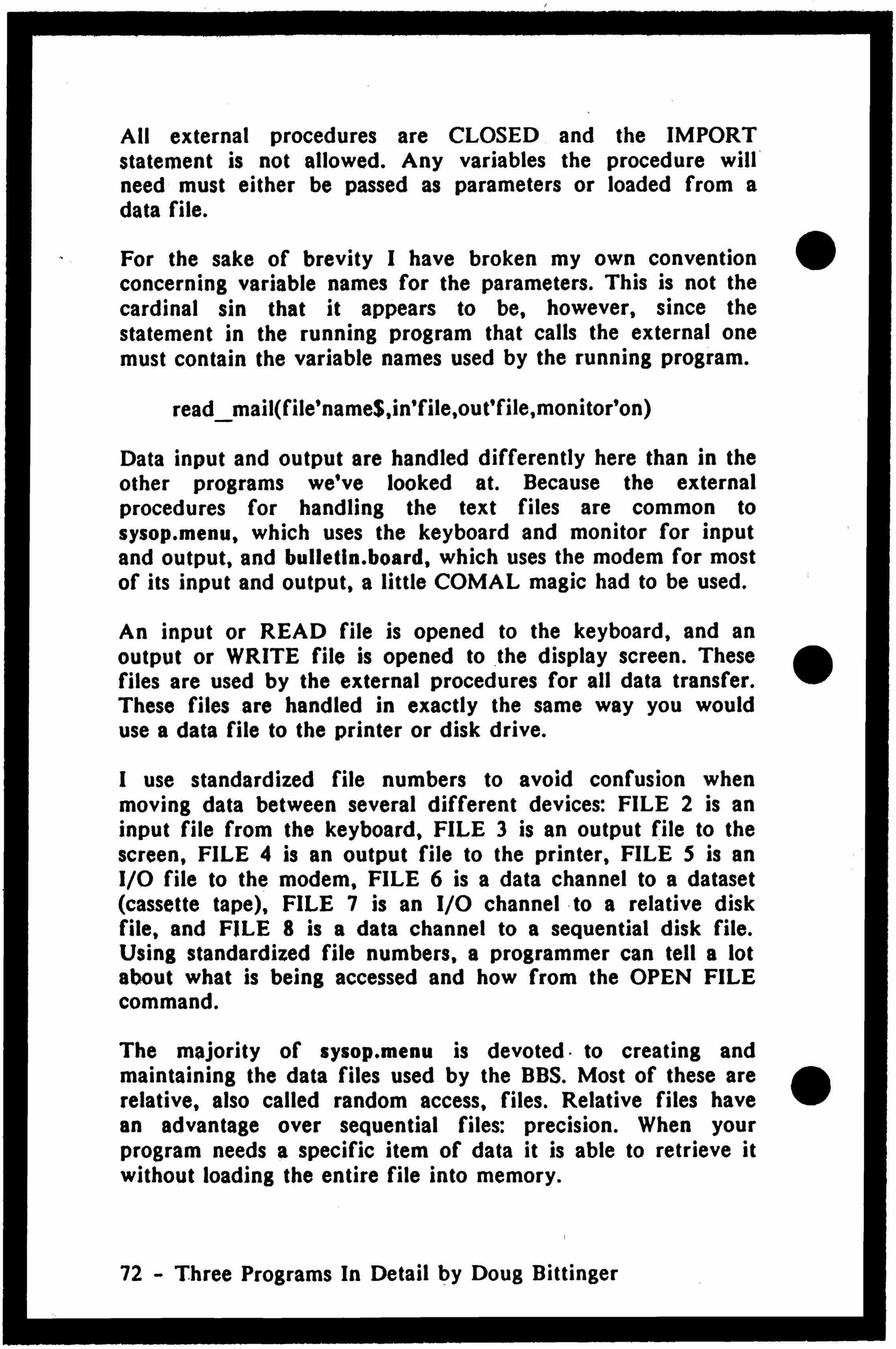


Figure 15. Grid design With H=20, V=12
Okidata Microline 193 @ 10 cpi
(shown actual printer size)

Printing your graph to a printer distorts the image as seen on the screen. The vertical axis has been compressed during the printing process. What you see on the screen is not what you get on the printer. However, you can control the visual spacing, both on the screen and on the printout, by knowing a few simple relationships.

The difference between screen and printer images is partially due to the fact that a pixel on the screen is not really a perfect circle, but is shaped more like a rectangle. On the Commodore 1702 video monitor your typical pixel is approximately 0.02875 inches



All external procedures are CLOSED and the IMPORT statement is not allowed. Any variables the procedure will need must either be passed as parameters or loaded from a data file.

For the sake of brevity I have broken my own convention concerning variable names for the parameters. This is not the cardinal sin that it appears to be, however, since the statement in the running program that calls the external one must contain the variable names used by the running program.

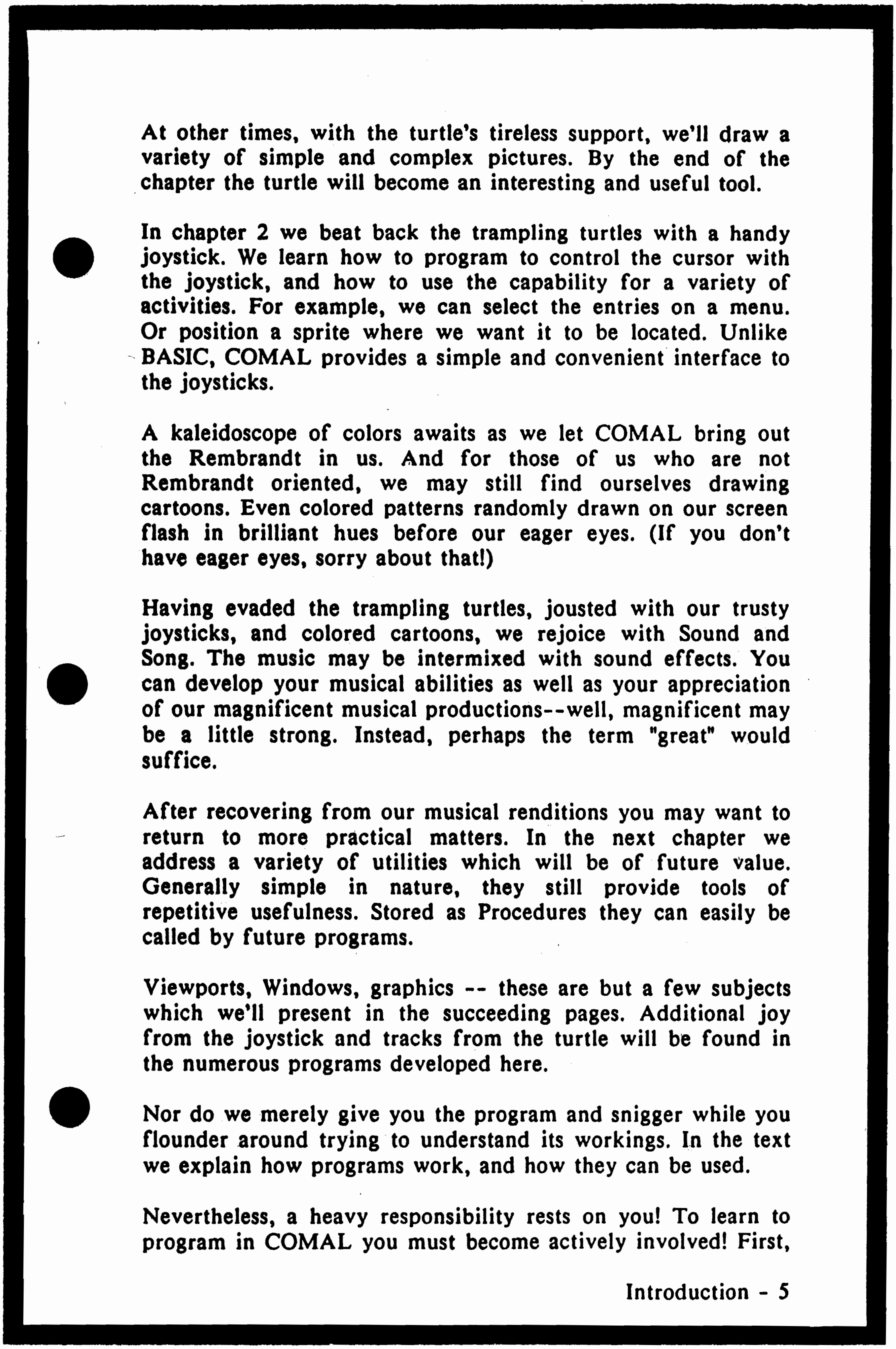
```
read_mail(file'name$,in'file,out'file,monitor'on)
```

Data input and output are handled differently here than in the other programs we've looked at. Because the external procedures for handling the text files are common to **sysop.menu**, which uses the keyboard and monitor for input and output, and **bulletin.board**, which uses the modem for most of its input and output, a little COMAL magic had to be used.

An input or READ file is opened to the keyboard, and an output or WRITE file is opened to the display screen. These files are used by the external procedures for all data transfer. These files are handled in exactly the same way you would use a data file to the printer or disk drive.

I use standardized file numbers to avoid confusion when moving data between several different devices: FILE 2 is an input file from the keyboard, FILE 3 is an output file to the screen, FILE 4 is an output file to the printer, FILE 5 is an I/O file to the modem, FILE 6 is a data channel to a dataset (cassette tape), FILE 7 is an I/O channel to a relative disk file, and FILE 8 is a data channel to a sequential disk file. Using standardized file numbers, a programmer can tell a lot about what is being accessed and how from the OPEN FILE command.

The majority of **sysop.menu** is devoted to creating and maintaining the data files used by the BBS. Most of these are relative, also called random access, files. Relative files have an advantage over sequential files: precision. When your program needs a specific item of data it is able to retrieve it without loading the entire file into memory.



At other times, with the turtle's tireless support, we'll draw a variety of simple and complex pictures. By the end of the chapter the turtle will become an interesting and useful tool.

In chapter 2 we beat back the trampling turtles with a handy joystick. We learn how to program to control the cursor with the joystick, and how to use the capability for a variety of activities. For example, we can select the entries on a menu. Or position a sprite where we want it to be located. Unlike BASIC, COMAL provides a simple and convenient interface to the joysticks.

A kaleidoscope of colors awaits as we let COMAL bring out the Rembrandt in us. And for those of us who are not Rembrandt oriented, we may still find ourselves drawing cartoons. Even colored patterns randomly drawn on our screen flash in brilliant hues before our eager eyes. (If you don't have eager eyes, sorry about that!)

Having evaded the trampling turtles, jousted with our trusty joysticks, and colored cartoons, we rejoice with Sound and Song. The music may be intermixed with sound effects. You can develop your musical abilities as well as your appreciation of our magnificent musical productions--well, magnificent may be a little strong. Instead, perhaps the term "great" would suffice.

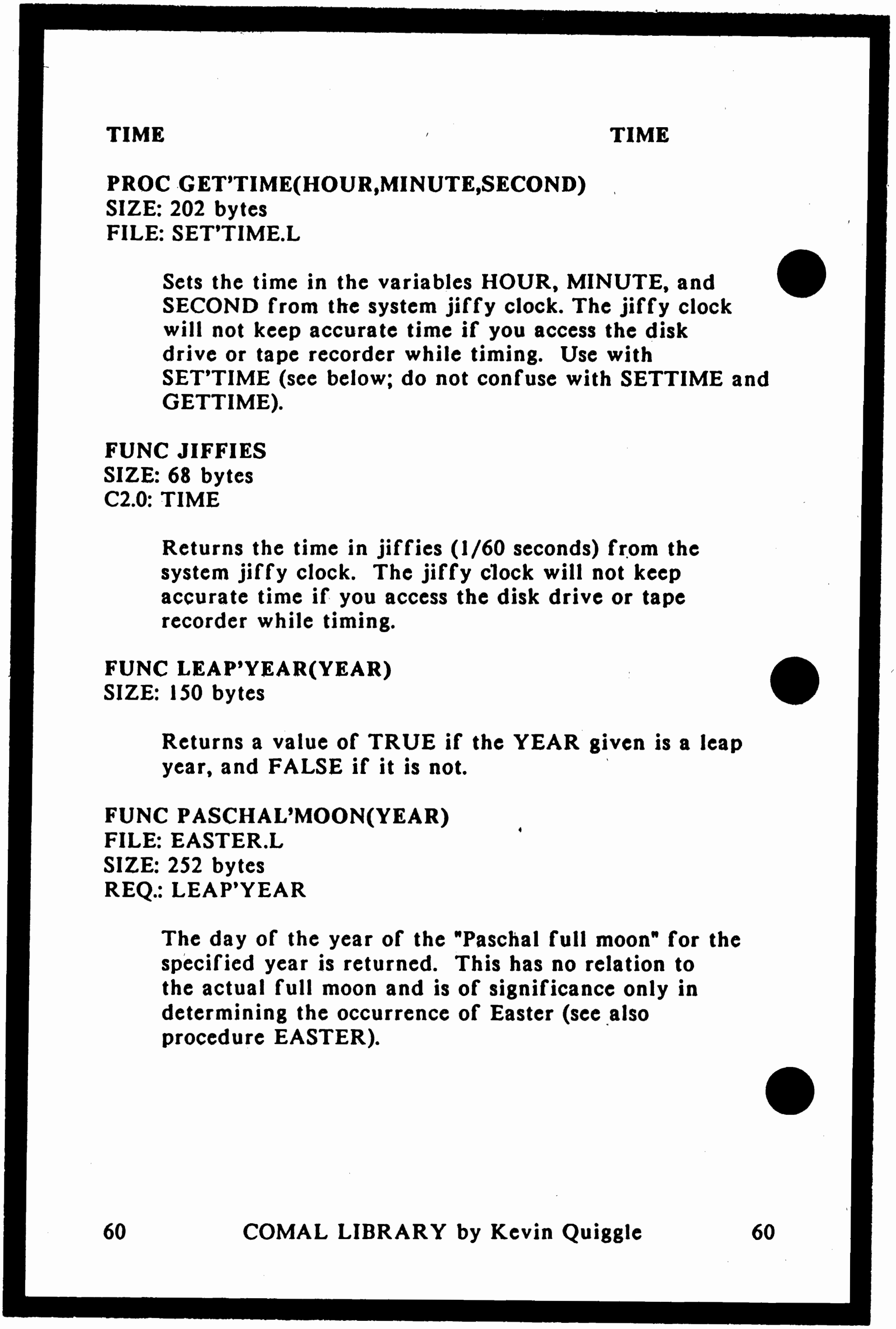
After recovering from our musical renditions you may want to return to more practical matters. In the next chapter we address a variety of utilities which will be of future value. Generally simple in nature, they still provide tools of repetitive usefulness. Stored as Procedures they can easily be called by future programs.

Viewports, Windows, graphics -- these are but a few subjects which we'll present in the succeeding pages. Additional joy from the joystick and tracks from the turtle will be found in the numerous programs developed here.

Nor do we merely give you the program and snigger while you flounder around trying to understand its workings. In the text we explain how programs work, and how they can be used.

Nevertheless, a heavy responsibility rests on you! To learn to program in COMAL you must become actively involved! First,

COMAL Collage by Frank and Melody Tymon, 168 pages - *programming*



PROC GET'TIME(HOUR,MINUTE,SECOND)
SIZE: 202 bytes
FILE: SET'TIME.L

Sets the time in the variables HOUR, MINUTE, and SECOND from the system jiffy clock. The jiffy clock will not keep accurate time if you access the disk drive or tape recorder while timing. Use with SET'TIME (see below; do not confuse with SETTIME and GETTIME).

FUNC JIFFIES
SIZE: 68 bytes
C2.0: TIME

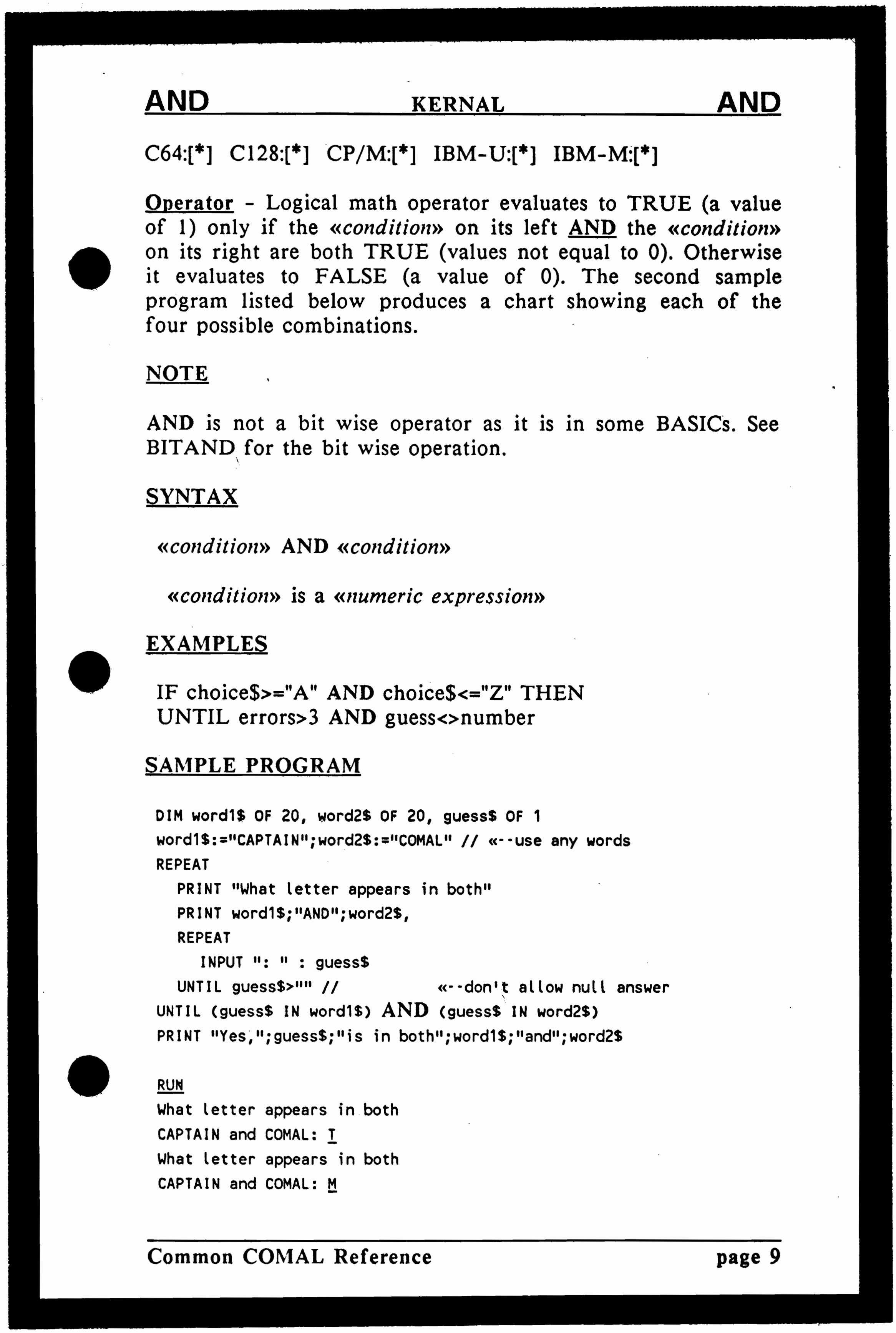
Returns the time in jiffies (1/60 seconds) from the system jiffy clock. The jiffy clock will not keep accurate time if you access the disk drive or tape recorder while timing.

FUNC LEAP'YEAR(YEAR)
SIZE: 150 bytes

Returns a value of TRUE if the YEAR given is a leap year, and FALSE if it is not.

FUNC PASCHAL'MOON(YEAR)
FILE: EASTER.L
SIZE: 252 bytes
REQ.: LEAP'YEAR

The day of the year of the "Paschal full moon" for the specified year is returned. This has no relation to the actual full moon and is of significance only in determining the occurrence of Easter (see also procedure EASTER).



AND KERNAL AND

C64:[*] C128:[*] CP/M:[*] IBM-U:[*] IBM-M:[*]

Operator - Logical math operator evaluates to TRUE (a value of 1) only if the «*condition*» on its left AND the «*condition*» on its right are both TRUE (values not equal to 0). Otherwise it evaluates to FALSE (a value of 0). The second sample program listed below produces a chart showing each of the four possible combinations.

NOTE

AND is not a bit wise operator as it is in some BASICs. See BITAND for the bit wise operation.

SYNTAX

«*condition*» AND «*condition*»

«*condition*» is a «*numeric expression*»

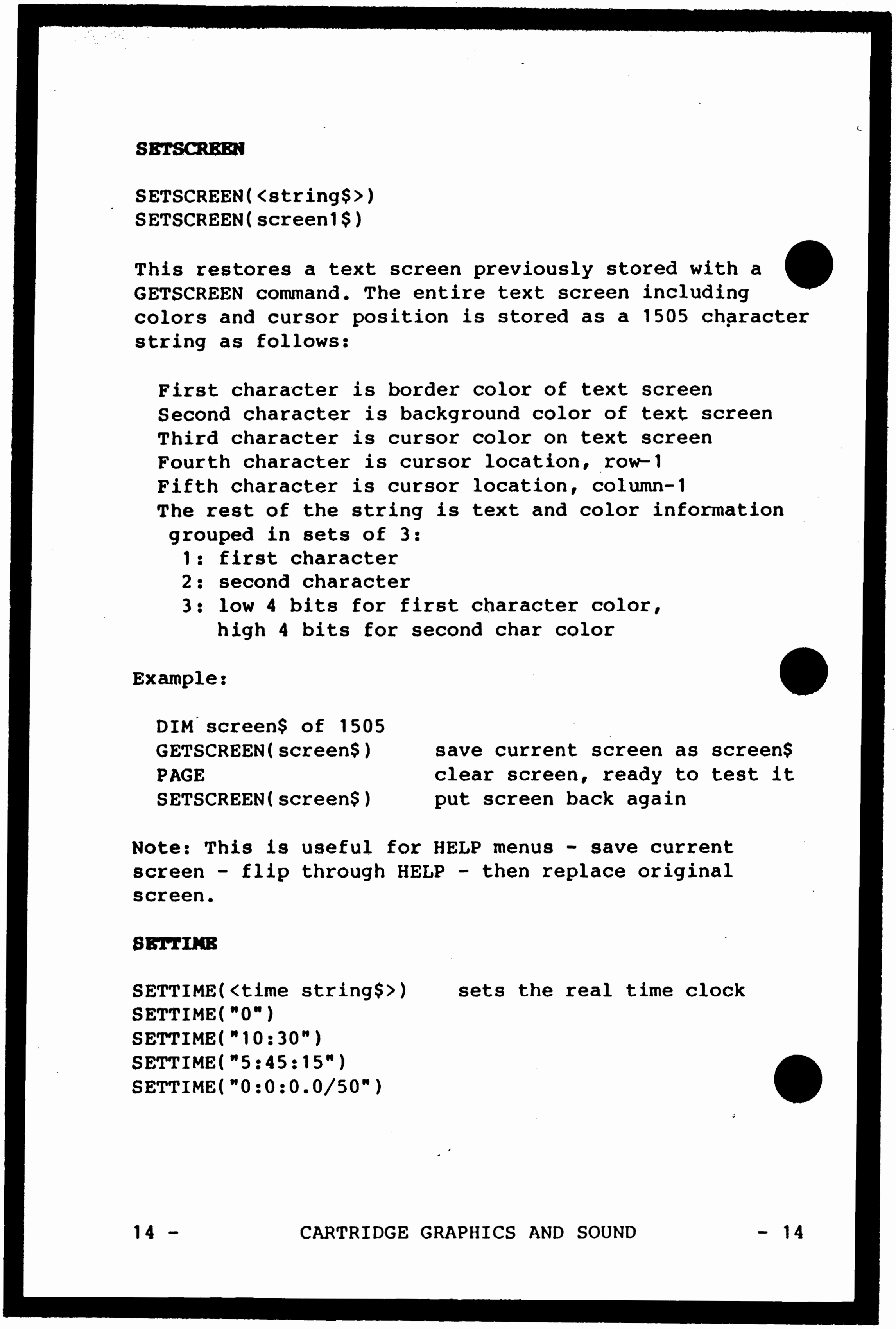
EXAMPLES

```
IF choice$>="A" AND choice$<="Z" THEN
UNTIL errors>3 AND guess<>number
```

SAMPLE PROGRAM

```
DIM word1$ OF 20, word2$ OF 20, guess$ OF 1
word1$:="CAPTAIN";word2$:="COMAL" // «--use any words
REPEAT
  PRINT "What letter appears in both"
  PRINT word1$;"AND";word2$,
  REPEAT
    INPUT ": " : guess$
  UNTIL guess$>"" //          «--don't allow null answer
UNTIL (guess$ IN word1$) AND (guess$ IN word2$)
PRINT "Yes,";guess$;"is in both";word1$;"and";word2$

RUN
What letter appears in both
CAPTAIN and COMAL: T
What letter appears in both
CAPTAIN and COMAL: M
```



SETSCREEN

```
SETSCREEN(<string$>)
SETSCREEN(screen1$)
```

This restores a text screen previously stored with a GETSCREEN command. The entire text screen including colors and cursor position is stored as a 1505 character string as follows:

```
  First character is border color of text screen
  Second character is background color of text screen
  Third character is cursor color on text screen
  Fourth character is cursor location, row-1
  Fifth character is cursor location, column-1
  The rest of the string is text and color information
   grouped in sets of 3:
    1: first character
    2: second character
    3: low 4 bits for first character color,
       high 4 bits for second char color
```

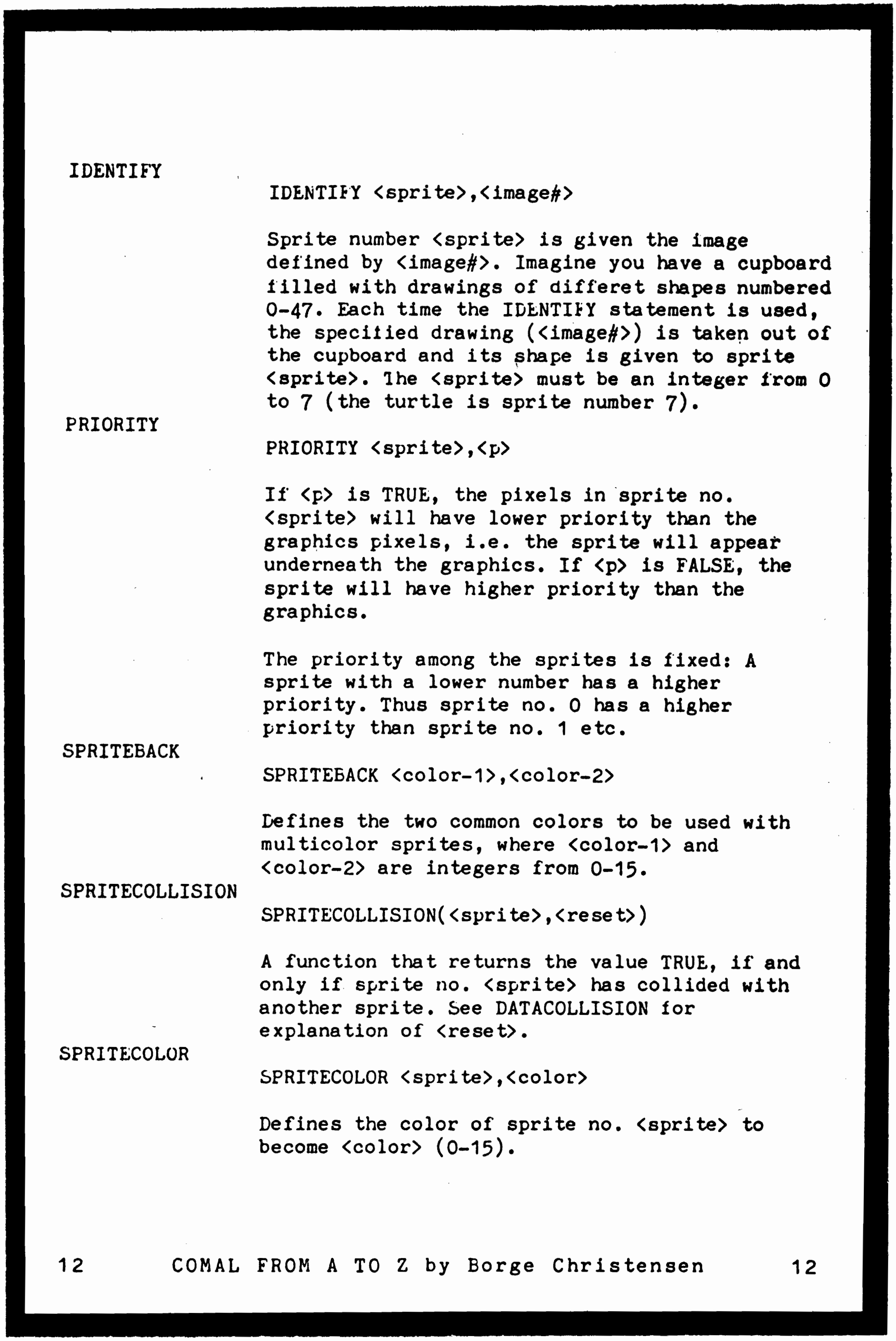
Example:

```
  DIM screen$ of 1505
  GETSCREEN(screen$)     save current screen as screen$
  PAGE                   clear screen, ready to test it
  SETSCREEN(screen$)     put screen back again
```

Note: This is useful for HELP menus - save current screen - flip through HELP - then replace original screen.

SETTIME

```
SETTIME(<time string$>)    sets the real time clock
SETTIME("0")
SETTIME("10:30")
SETTIME("5:45:15")
SETTIME("0:0:0.0/50")
```



IDENTIFY

IDENTIFY <sprite>,<image#>

Sprite number <sprite> is given the image defined by <image#>. Imagine you have a cupboard filled with drawings of differet shapes numbered 0-47. Each time the IDENTIFY statement is used, the specified drawing (<image#>) is taken out of the cupboard and its shape is given to sprite <sprite>. The <sprite> must be an integer from 0 to 7 (the turtle is sprite number 7).

PRIORITY

PRIORITY <sprite>,<p>

If <p> is TRUE, the pixels in sprite no. <sprite> will have lower priority than the graphics pixels, i.e. the sprite will appear underneath the graphics. If <p> is FALSE, the sprite will have higher priority than the graphics.

The priority among the sprites is fixed: A sprite with a lower number has a higher priority. Thus sprite no. 0 has a higher priority than sprite no. 1 etc.

SPRITEBACK

SPRITEBACK <color-1>,<color-2>

Defines the two common colors to be used with multicolor sprites, where <color-1> and <color-2> are integers from 0-15.

SPRITECOLLISION

SPRITECOLLISION(<sprite>,<reset>)

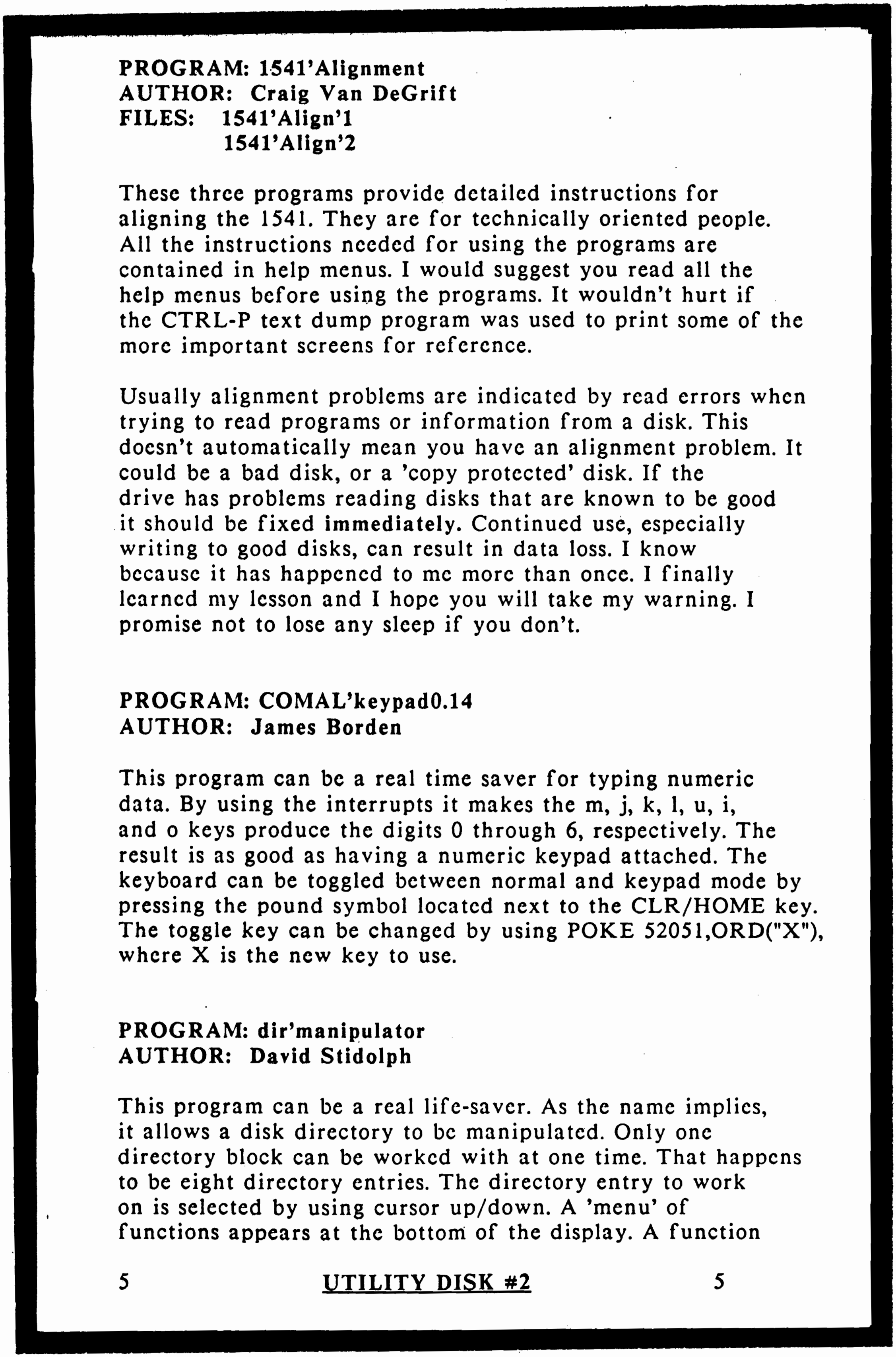
A function that returns the value TRUE, if and only if sprite no. <sprite> has collided with another sprite. See DATACOLLISION for explanation of <reset>.

SPRITECOLOR

SPRITECOLOR <sprite>,<color>

Defines the color of sprite no. <sprite> to become <color> (0-15).

12 COMAL FROM A TO Z by Borge Christensen 12



PROGRAM: 1541'Alignment
AUTHOR: Craig Van DeGrift
FILES: 1541'Align'1
1541'Align'2

These three programs provide detailed instructions for aligning the 1541. They are for technically oriented people. All the instructions needed for using the programs are contained in help menus. I would suggest you read all the help menus before using the programs. It wouldn't hurt if the CTRL-P text dump program was used to print some of the more important screens for reference.

Usually alignment problems are indicated by read errors when trying to read programs or information from a disk. This doesn't automatically mean you have an alignment problem. It could be a bad disk, or a 'copy protected' disk. If the drive has problems reading disks that are known to be good it should be fixed **immediately**. Continued use, especially writing to good disks, can result in data loss. I know because it has happened to me more than once. I finally learned my lesson and I hope you will take my warning. I promise not to lose any sleep if you don't.

PROGRAM: COMAL'keypad0.14
AUTHOR: James Borden

This program can be a real time saver for typing numeric data. By using the interrupts it makes the m, j, k, l, u, i, and o keys produce the digits 0 through 6, respectively. The result is as good as having a numeric keypad attached. The keyboard can be toggled between normal and keypad mode by pressing the pound symbol located next to the CLR/HOME key. The toggle key can be changed by using POKE 52051,ORD("X"), where X is the new key to use.

PROGRAM: dir'manipulator
AUTHOR: David Stidolph

This program can be a real life-saver. As the name implies, it allows a disk directory to be manipulated. Only one directory block can be worked with at one time. That happens to be eight directory entries. The directory entry to work on is selected by using cursor up/down. A 'menu' of functions appears at the bottom of the display. A function

Utility Disk #2 22 pages - *utilities and printer dumps*

NUMBERS!

To deal with this, we use the command

```
            randomize
```

Using the TIME function built into the computer (which works on "jiffies" --60 jiffies in 1 second) RANDOMIZE initializes ("seeds") the first random number. Then when we call for the random number, we get a true one.

This program will print one "throw" of a pair of dice.

```
1000 page
1010
1020 // seeding the random number generator
1030
1040 randomize
1050
1060 // a pair of dice
1070
1070 die1:=rnd(1,6)
1080 die2:=rnd(1,6)
1090
1100 dice:=die1 + die2
1110
1120 print dice
```

When RUN, the program will generate a number from 2 through 12 --perhaps, the first time, a 10.

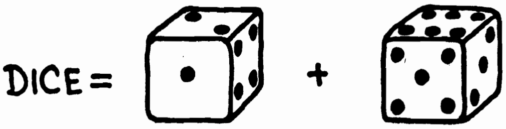
RUN the program several times, noting the printouts.

The beginning and ending numbers in the range can differ in value. If we wish to generate random numbers from 39 to 72, we enter

```
            print rnd(39,72)
```

for 117 through 209

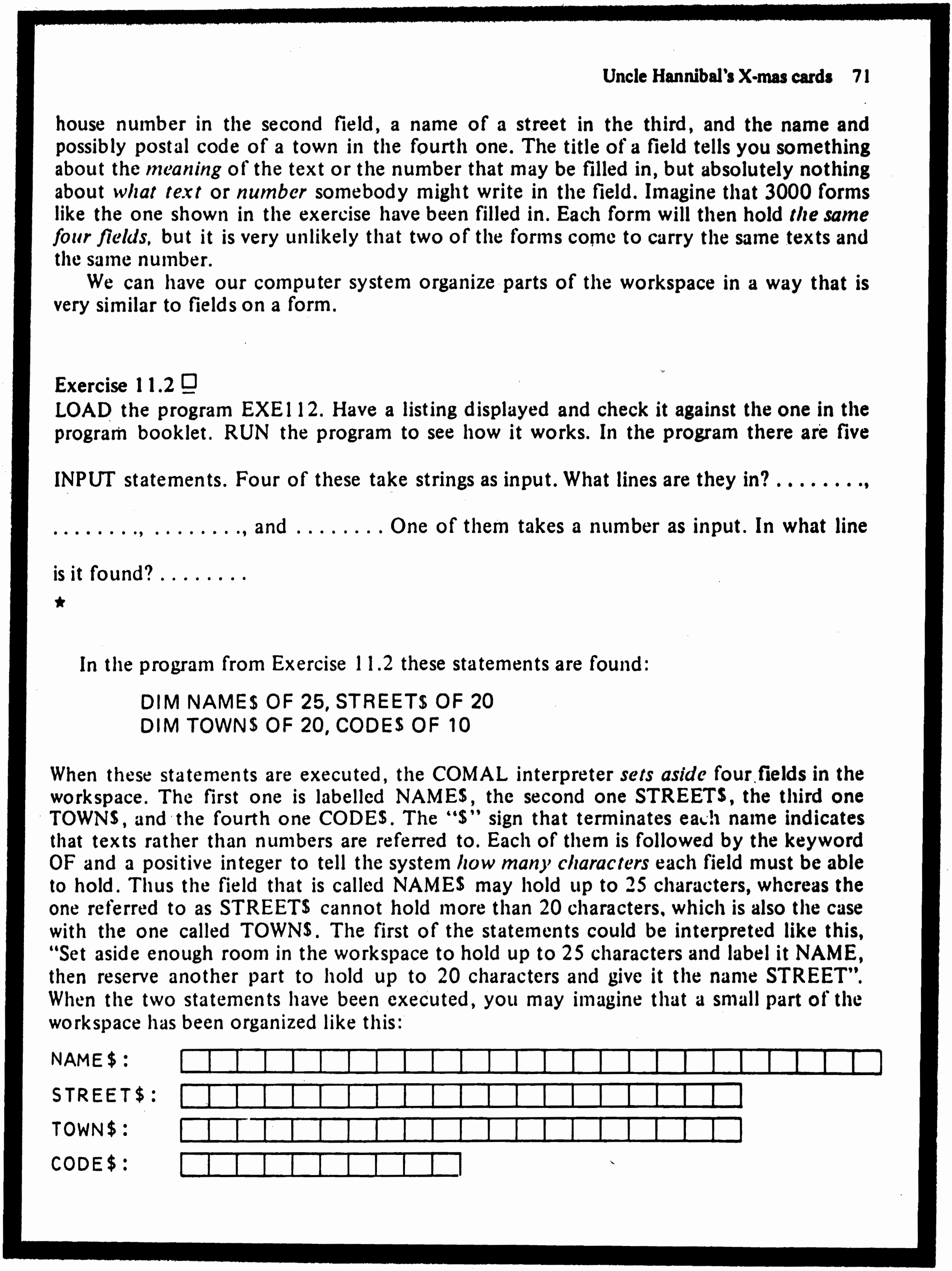
```
            print rnd(117,209)
```



Quite straight forward!

DICE:= RND(1,6) + RND(1,6)

DICE:= DIE1 + DIE2



house number in the second field, a name of a street in the third, and the name and possibly postal code of a town in the fourth one. The title of a field tells you something about the *meaning* of the text or the number that may be filled in, but absolutely nothing about *what text* or *number* somebody might write in the field. Imagine that 3000 forms like the one shown in the exercise have been filled in. Each form will then hold ***the same four fields***, but it is very unlikely that two of the forms come to carry the same texts and the same number.

We can have our computer system organize parts of the workspace in a way that is very similar to fields on a form.

Exercise 11.2 □

LOAD the program EXE112. Have a listing displayed and check it against the one in the program booklet. RUN the program to see how it works. In the program there are five

INPUT statements. Four of these take strings as input. What lines are they in? ....,

........, ..., and .... One of them takes a number as input. In what line

is it found? ....

★

In the program from Exercise 11.2 these statements are found:

```
DIM NAME$ OF 25, STREET$ OF 20
DIM TOWN$ OF 20, CODE$ OF 10
```

When these statements are executed, the COMAL interpreter *sets aside* four fields in the workspace. The first one is labelled NAME$, the second one STREET$, the third one TOWN$, and the fourth one CODE$. The "$" sign that terminates each name indicates that texts rather than numbers are referred to. Each of them is followed by the keyword OF and a positive integer to tell the system *how many characters* each field must be able to hold. Thus the field that is called NAME$ may hold up to 25 characters, whereas the one referred to as STREET$ cannot hold more than 20 characters, which is also the case with the one called TOWN$. The first of the statements could be interpreted like this, "Set aside enough room in the workspace to hold up to 25 characters and label it NAME, then reserve another part to hold up to 20 characters and give it the name STREET". When the two statements have been executed, you may imagine that a small part of the workspace has been organized like this:

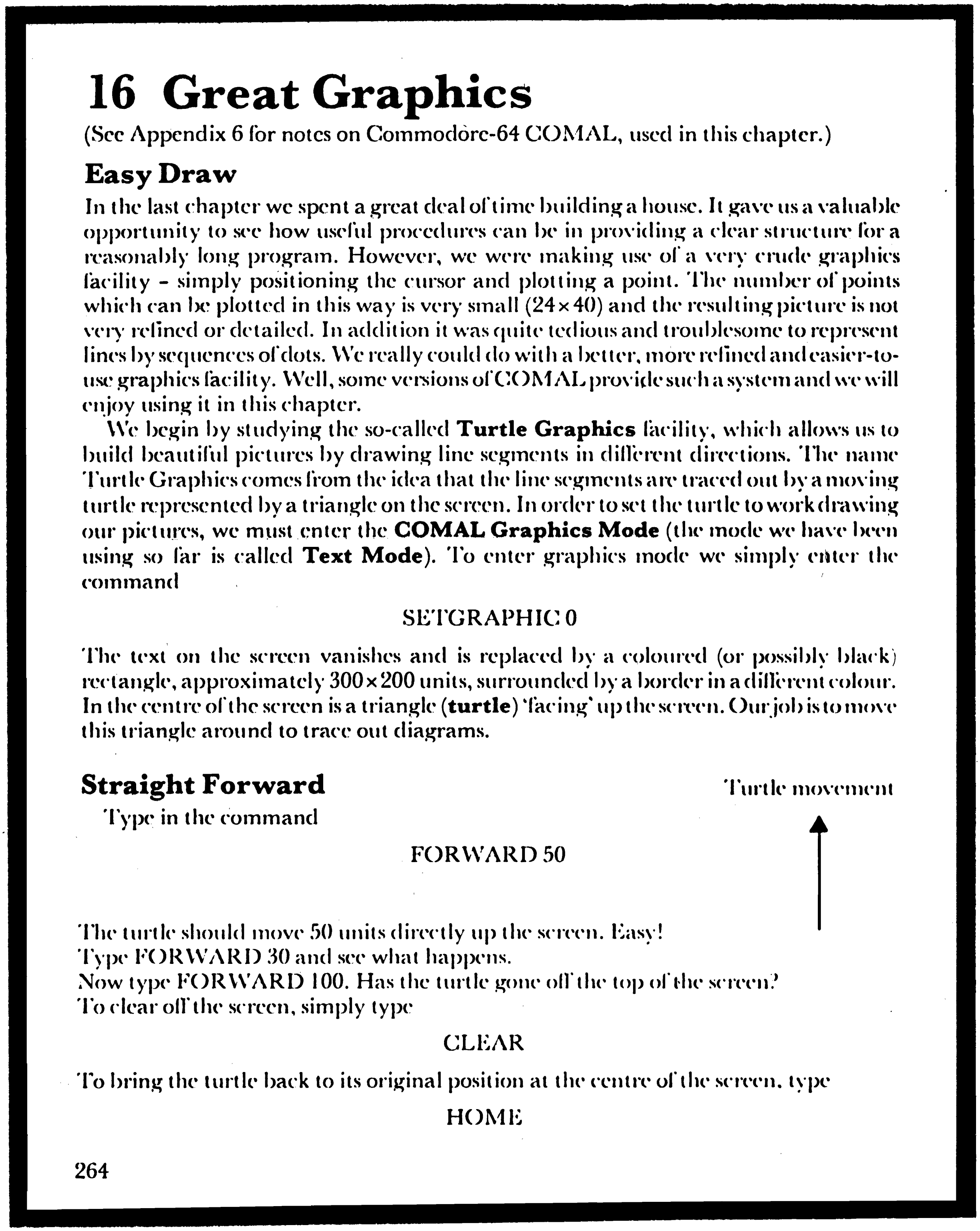
NAME$:

STREET$:

TOWN$:

CODE$:

Beginning COMAL by Borge Christensen, 333 pages - *textbook*



16 Great Graphics

(See Appendix 6 for notes on Commodore-64 COMAL, used in this chapter.)

Easy Draw

In the last chapter we spent a great deal of time building a house. It gave us a valuable opportunity to see how useful procedures can be in providing a clear structure for a reasonably long program. However, we were making use of a very crude graphics facility - simply positioning the cursor and plotting a point. The number of points which can be plotted in this way is very small (24×40) and the resulting picture is not very refined or detailed. In addition it was quite tedious and troublesome to represent lines by sequences of dots. We really could do with a better, more refined and easier-to-use graphics facility. Well, some versions of COMAL provide such a system and we will enjoy using it in this chapter.

We begin by studying the so-called **Turtle Graphics** facility, which allows us to build beautiful pictures by drawing line segments in different directions. The name Turtle Graphics comes from the idea that the line segments are traced out by a moving turtle represented by a triangle on the screen. In order to set the turtle to work drawing our pictures, we must enter the **COMAL Graphics Mode** (the mode we have been using so far is called **Text Mode**). To enter graphics mode we simply enter the command

```
SETGRAPHIC 0
```

The text on the screen vanishes and is replaced by a coloured (or possibly black) rectangle, approximately 300×200 units, surrounded by a border in a different colour. In the centre of the screen is a triangle (**turtle**) 'facing' up the screen. Our job is to move this triangle around to trace out diagrams.

Straight Forward

Type in the command

```
FORWARD 50
```

Turtle movement ↑

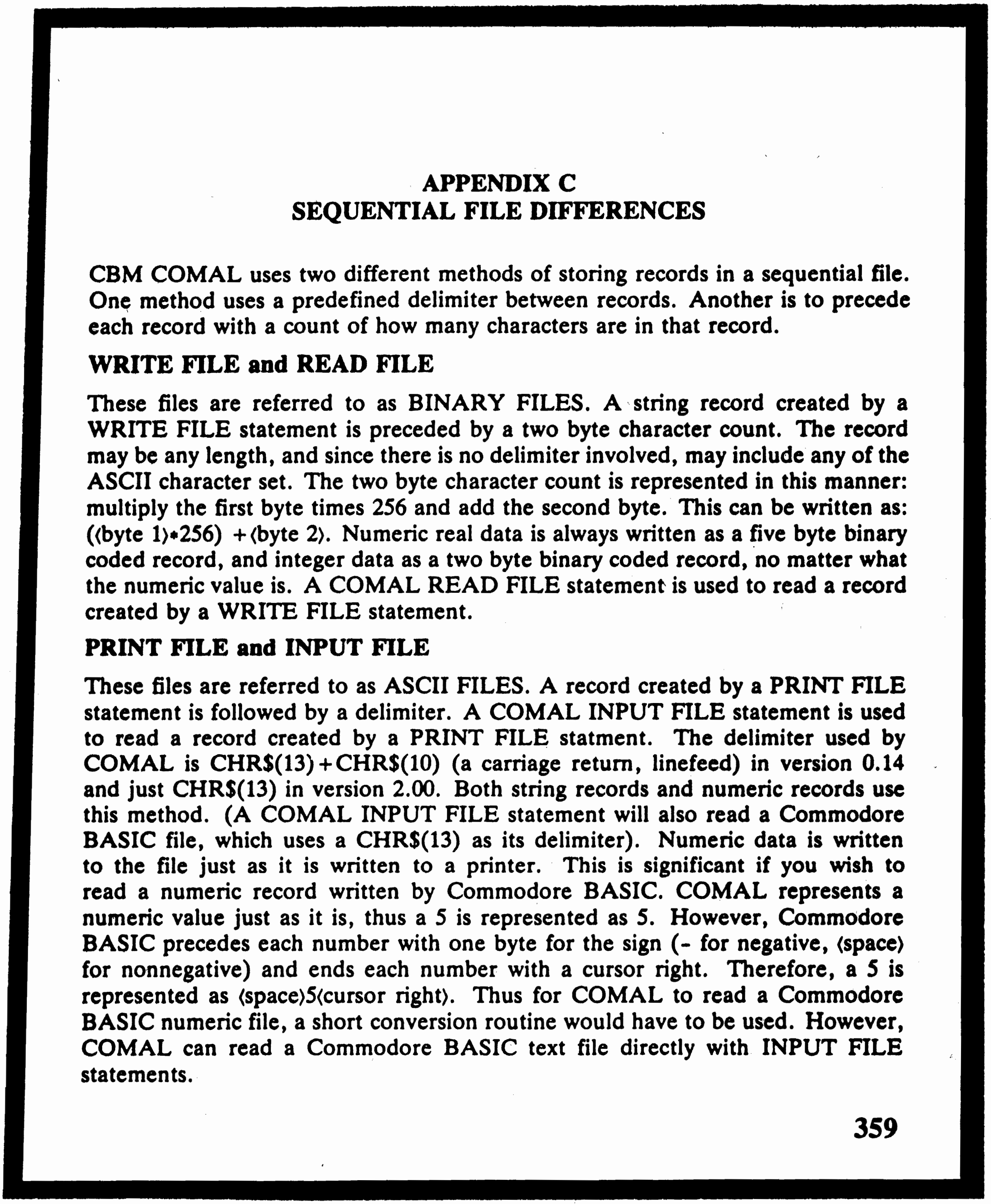
The turtle should move 50 units directly up the screen. Easy!
Type FORWARD 30 and see what happens.
Now type FORWARD 100. Has the turtle gone off the top of the screen?
To clear off the screen, simply type

```
CLEAR
```

To bring the turtle back to its original position at the centre of the screen, type

```
HOME
```

Foundations with COMAL by John Kelly, 363 pages - *textbook*



APPENDIX C
SEQUENTIAL FILE DIFFERENCES

CBM COMAL uses two different methods of storing records in a sequential file. One method uses a predefined delimiter between records. Another is to precede each record with a count of how many characters are in that record.

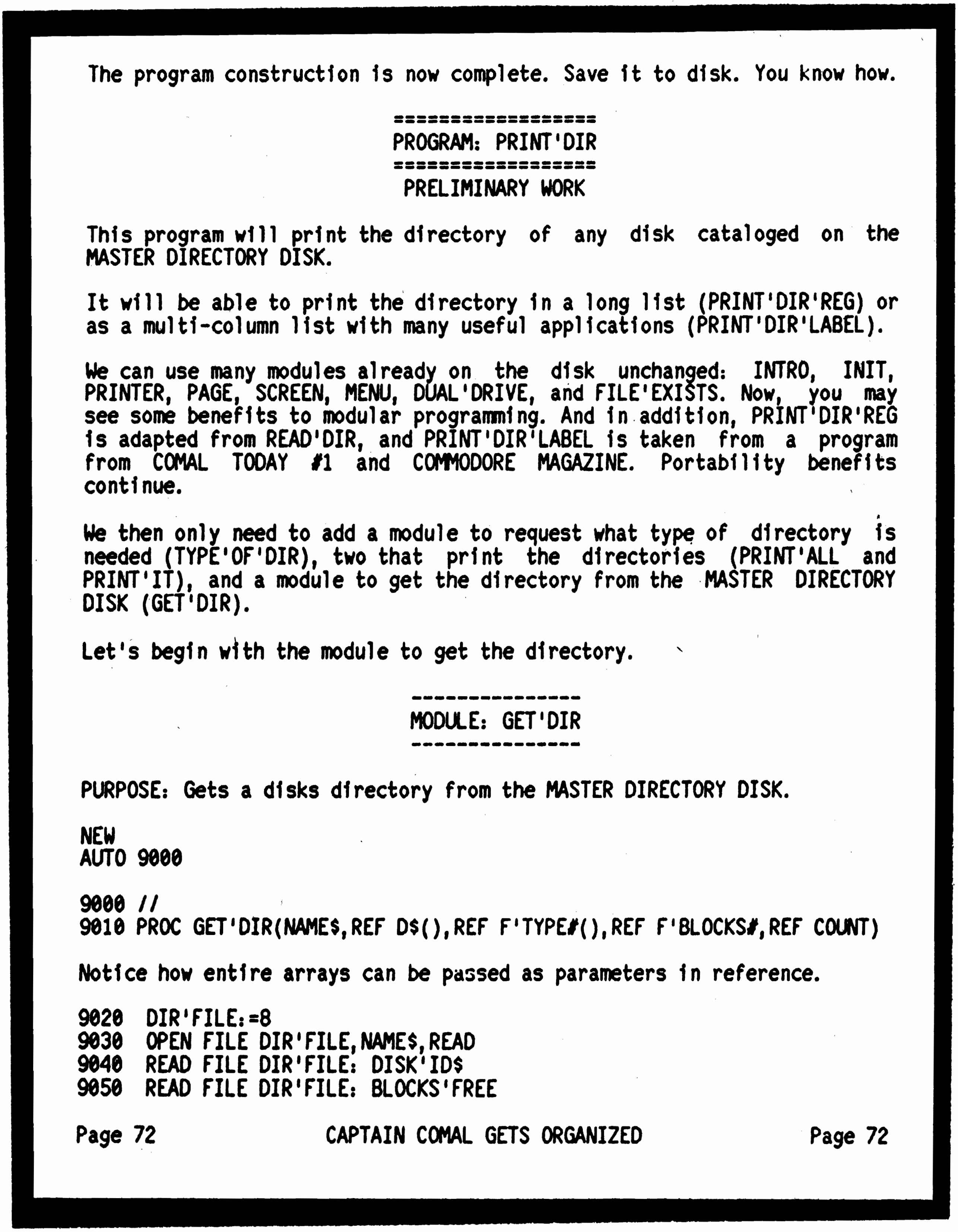
WRITE FILE and READ FILE

These files are referred to as BINARY FILES. A string record created by a WRITE FILE statement is preceded by a two byte character count. The record may be any length, and since there is no delimiter involved, may include any of the ASCII character set. The two byte character count is represented in this manner: multiply the first byte times 256 and add the second byte. This can be written as: (⟨byte 1⟩*256) +⟨byte 2⟩. Numeric real data is always written as a five byte binary coded record, and integer data as a two byte binary coded record, no matter what the numeric value is. A COMAL READ FILE statement is used to read a record created by a WRITE FILE statement.

PRINT FILE and INPUT FILE

These files are referred to as ASCII FILES. A record created by a PRINT FILE statement is followed by a delimiter. A COMAL INPUT FILE statement is used to read a record created by a PRINT FILE statment. The delimiter used by COMAL is CHR$(13)+CHR$(10) (a carriage return, linefeed) in version 0.14 and just CHR$(13) in version 2.00. Both string records and numeric records use this method. (A COMAL INPUT FILE statement will also read a Commodore BASIC file, which uses a CHR$(13) as its delimiter). Numeric data is written to the file just as it is written to a printer. This is significant if you wish to read a numeric record written by Commodore BASIC. COMAL represents a numeric value just as it is, thus a 5 is represented as 5. However, Commodore BASIC precedes each number with one byte for the sign (- for negative, ⟨space⟩ for nonnegative) and ends each number with a cursor right. Therefore, a 5 is represented as ⟨space⟩5⟨cursor right⟩. Thus for COMAL to read a Commodore BASIC numeric file, a short conversion routine would have to be used. However, COMAL can read a Commodore BASIC text file directly with INPUT FILE statements.

COMAL Handbook by Len Lindsay, 479 pages, - *reference*



The program construction is now complete. Save it to disk. You know how.

==================
PROGRAM: PRINT'DIR
==================

PRELIMINARY WORK

This program will print the directory of any disk cataloged on the MASTER DIRECTORY DISK.

It will be able to print the directory in a long list (PRINT'DIR'REG) or as a multi-column list with many useful applications (PRINT'DIR'LABEL).

We can use many modules already on the disk unchanged: INTRO, INIT, PRINTER, PAGE, SCREEN, MENU, DUAL'DRIVE, and FILE'EXISTS. Now, you may see some benefits to modular programming. And in addition, PRINT'DIR'REG is adapted from READ'DIR, and PRINT'DIR'LABEL is taken from a program from COMAL TODAY #1 and COMMODORE MAGAZINE. Portability benefits continue.

We then only need to add a module to request what type of directory is needed (TYPE'OF'DIR), two that print the directories (PRINT'ALL and PRINT'IT), and a module to get the directory from the MASTER DIRECTORY DISK (GET'DIR).

Let's begin with the module to get the directory.

MODULE: GET'DIR

PURPOSE: Gets a disks directory from the MASTER DIRECTORY DISK.

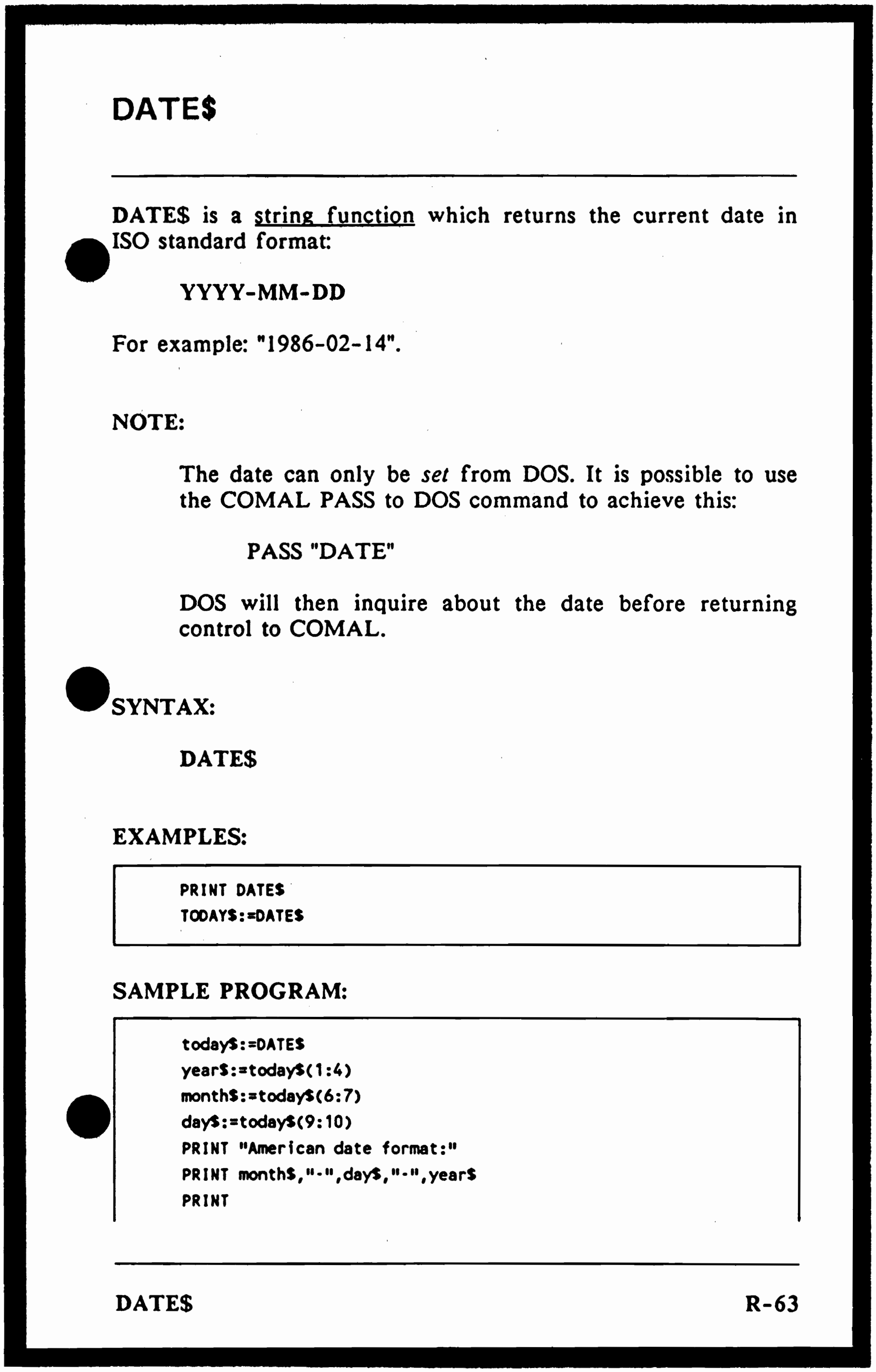
```
NEW
AUTO 9000

9000 //
9010 PROC GET'DIR(NAME$,REF D$(),REF F'TYPE#(),REF F'BLOCKS#,REF COUNT)
```

Notice how entire arrays can be passed as parameters in reference.

```
9020  DIR'FILE:=8
9030  OPEN FILE DIR'FILE,NAME$,READ
9040  READ FILE DIR'FILE: DISK'ID$
9050  READ FILE DIR'FILE: BLOCKS'FREE
```

Captain COMAL Gets Organized by Len Lindsay, 102 pages - *tutorial*



DATE$

DATE$ is a string function which returns the current date in ISO standard format:

YYYY-MM-DD

For example: "1986-02-14".

NOTE:

The date can only be *set* from DOS. It is possible to use the COMAL PASS to DOS command to achieve this:

PASS "DATE"

DOS will then inquire about the date before returning control to COMAL.

SYNTAX:

DATE$

EXAMPLES:

```
PRINT DATE$
TODAY$:=DATE$
```

SAMPLE PROGRAM:

```
today$:=DATE$
year$:=today$(1:4)
month$:=today$(6:7)
day$:=today$(9:10)
PRINT "American date format:"
PRINT month$,"-",day$,"-",year$
PRINT
```

Cartridge Tutorial Binder book 10-67
cartridge tutorial, COMAL 6-3,6-6,6-8
CASE and ON/GOTO 11-72
case characters, PLOTTEXT for lower 3-17
CASE command 1-6
case in COMAL 1-4
case, LIST in upper and lower 3-35,9-6,10-6
case mode lock 4-37,6-41,7-78
case mode, switching 7-78
case mode unlock 6-41
case PLOTTEXT in COMAL 0.14, improved lower 6-54
case PLOTTEXT in COMAL 0.14, lower 6-43
case printing, lower 2-28
case program listing, COMAL 0.14 mixed 5-22
CASE tutorial 11-4
cassette and COMAL, tape 1-19,1-25
CASTLE education system 2-24
CAT and DIR commands 6-39
CAT and IEEE printer 3-35
CAT bug, SELECT "LP:" and 1-25
CAT command tutorial 11-22
CAT command 6-35,10-26
CAT listing pause for COMAL 0.14 5-31
catalog : see also data base, inventory, mail list
catalog, COMAL programs 5-11
CHAIN command 1-6
CHANGE command 6-7
change'dev procedure 8-43
changing a function in a running program 10-64
changing C128 clock speed 10-29
changing disk drive device number 8-43,10-42
changing graphics turtle image 5-33
changing screen colors 10-52
changing sprite images 9-30
changing turtle coordinates 3-24
character codes in COMAL, ASCII 8-61
character generator, accessing RAM under I/O and 3-43
character limit, function key 32 7-77
character PEEK value 2-31
character ROM access from COMAL 2.0 6-37
character set, 80 column 7-54
character to screen, POKE 2-31
characters, Chinese 5-18
characters, custom 12-79
characters permitted in variable names 2-12
characters, PLOTTEXT for graphics 3-17
characters, PLOTTEXT for lower case 3-17
characters, PLOTTEXT for reverse 3-17
characters, PLOTTEXT reverse field 4-27
chart, pie 3-5
chart, turtle graphics 1-13
check, BASIC/COMAL LOAD 6-44
"check cartridge" program 8-6
checkbook register 12-31
CHECK'CARTRIDGE program bug 7-72
checking disk blocks free 10-27
checking program bug, cartridge 7-72
checking program, COMAL cartridge 6-40
checking remaining memory 11-38
chess problem, Eight Queens 5-11
Chinese characters 5-18
chip, C128 80 column video 12-3
chip, custom EPROM 12-3
CHIP for the COMAL 2.0 cartridge, QUICK 8-1
chips in the cartridge (hardware) 7-55
Christensen Annual Prize, Borge 11-71
CIE IEEE interface, cartridge and 5-35
CIE IEEE interface, COMAL and 3-4
CIE IEEE interface support code, relocating 3-34
CIE interface 1-26
CIRCLE simulation routine 3-24
circles, round 12-79
circuit, autocorrelator 5-39
circuit emulator, logic 5-39
circular clock program 4-29
clarified, COMAL versions 1-1
clear INPUT AT field 11-25
clear sprite collision flags 2-30
CLEAR 10-50
CLEAR'COLLISIONS procedure 2-30
clearing the keyboard buffer 5-47
clear'keys procedure 5-47
clock demo program, digital 5-20
clock, interrupt driven alarm 5-38
clock program, circular 4-29
clock program, machine language 5-26
clock program, sprite digital 3-43
clock speed, changing C128 10-29
clock, time-of-day 5-38
clock, TOD 5-38
closing disk files 11-25
CLOUD'FLUX.LIST graphing program example 7-48
clouds, interstellar dust 7-48
Club, starting a COMAL 2-22,10-48
Coast Commodore Show (humor), West 6-18
code from disk, load machine 6-82
code procedures, Epson FX-80 printer 6-65,9-66
code, relocating CIE IEEE interface support 3-34
code table, Commodore ASCII 9-20
code to DATA statements, convert machine 7-60
code trainer, HAM radio 9-48
codes in COMAL, ASCII character 8-61
codes, key display 9-22
codes, modified ASCII Commodore 7-78
codes, SELECT and printer control 2-28
codes to printer, sending escape 6-49
College Commodore Computer Camp, Lincoln 7-29,8-1
collision detection, sprite 10-24

EXERCISE 6: PROCEDURES FOR MODULARITY

Lesson 6 in THE COMAL COMPUTER TUTOR [TUTORIAL DISK] should be completed before starting this exercise. After you have entered [from COMAL] the NEW command and the AUTO command, type in this short program:

```
GET'NUMBERS
FIND'AVERAGE
PRINT'RESULTS
```

Get out of the automatic line-numbering system by tapping the RETURN key a second time, RUN this incomplete program, and write the ERROR MESSAGE that appears on the screen:

____

____

This is the typical error message that appears when you try to "call" or execute a procedure that doesn't exist. You already know that the procedure itself is just a list of statements grouped together to get a specific job done. Enter AUTO 4Ø and continue with:

```
//
PROC GET'NUMBERS
 INPUT "ENTER FIRST NUMBER: ":FIRST
 INPUT "ENTER SECOND NUMBER: ":SECOND
ENDPROC
```

Recall that the slash marks (//) in line 4Ø help to separate this first procedure from the statements that make up the "main" program. It will make the program listing easier to read.

Tap the RETURN key a second time to get out of the AUTO system and RUN the program. What question appears on the screen?

____

Go ahead and enter a number. What number did you enter?

____

When you press RETURN to enter this last number, a familiar ERROR MESSAGE should appear. Notice that the error message tells you where it found the error -- not at the line that calls the first procedure (there was nothing wrong with it). Rather, the error is at line 2Ø. Important point: The computer carries out the statements of the main program, one at a time, until it reaches the last statement -- nothing new here. The first statement told the computer to "do" procedure GET'NUMBERS. You notice that this part of the program went just fine. The next statement of the main program was to do the procedure named FIND'AVERAGE. The computer couldn't find this procedure (because we have written its statements yet) and so the program stopped here with the familiar error message.

Fourth Private Sale

SALE ORDER FORM

Prepaid US dollars only. Subscriber#____________

Name:______________________________

Street:______________________________

City/St/Zip:__________________________

Visa/MC#:____________________________

Exp Date:______ Signature:________________

Prices subject to change without notice.

This is our fourth private sale. But this time it is even bigger and we are doing it just before Christmas. When someone asks you what you need for Christmas, be specific!

This time we are including the sale flyer as an insert in *COMAL Today* rather than a separate mailer. This gives us a four page pullout, rather than a two page flyer, like the last three times. As usual, the sale is for a very limited time, ending on Thanksgiving Day (Nov 26).

TO ORDER:

- Pull out this 4 page section (pgs 39-42)
- Fill in subscriber# / address (above)
- Check [x] each item you want to order
- Add up total for items (fill in below)
- Add shipping/handling (fill in below)
- Send check/money order for Total
  -- OR -- Fill in charge info (above) ...
  we will calculate the total & shipping
- Mail in the order... or call 608-222-4432

NOTICE: Minimum order $10. Shipping is extra at usual rates: $2 minimum shipping, $3 per book, Canada & First Class add $1 more per book and newsletter issue.

ENTER TOTALS HERE:

Total for items ordered: $____________

Total for shipping: $____________

Grand Total enclosed: $____________

SUBSCRIPTION SPECIALS

If you are an expired subscriber (last issue is before #20), **you must renew your subscription** before you can take advantage of these sale prices (we automatically start your renewal with the issue where you left off, to keep your *COMAL Today* collection complete). New subscribers can order sale items at the same time as subscribing.

If you share a *COMAL Today* newsletter or disk subscription with somebody, this is your chance to show your support for our efforts. Get your own subscription at the lowest rates ever!

[ ] $10.95 - 5 issue subscription/renewal
(Canada add $1 per issue for 1st Class)

[ ] $24.95 - 5 disk subscription/renewal

[ ] $1 each backissues 5-12:
Circle issues wanted: 5 6 7 8 9 10 11 12

[ ] $2.50 each backissues 13-19:
Circle issues wanted: 13 14 15 16 17 18 19

Early Days

We thought they were gone, but we just found some original first, second and fourth issues of *COMAL Today*! Not reprints! First come first served! Order two. A real collectors item!

[ ] **COMAL Today issue #1 - $3.95**

[ ] **COMAL Today issue #2 - $2.95**

[ ] **COMAL Today issue #4 - $2.50**

VOTE HERE

1) **What are your favorite articles in this issue?**
pages __________ & __________

2) **Check the COMAL(s) that you use:**
[ ] C64 0.14 [ ] C64 2.0 Cart [ ] Superchip
[ ] C128 2.0 Cart [ ] CP/M COMAL
[ ] UniComal IBM PC COMAL

3) **What computer not on the above list do you wish there were a COMAL for?**__________

more»

FREE DISKS

Take a **free** COMAL disk for each item you buy! Buy a book, get a free disk. Buy the Power Box, get a free disk. Renew your subscription, get a free disk. Even buy a disk, get another disk free! To buy a disk from the list, write the word buy next to the [ ] box. We reduced the price to $7.95 each for this sale. **Choose from any of the MANY disks below:**

[ ] CP/M COMAL Demo Disk ($4)
[ ] Beginning COMAL disk §
[ ] Foundations with COMAL disk §
[ ] COMAL Handbook disk §
[ ] COMAL Today INDEX disk §
[ ] Data Base disk (0.14 & 2.0)
[ ] Font Disk #1 (0.14 & 2.0)
[ ] Games Disk #1 (0.14 & 2.0)
[ ] Modem Disk (0.14 & 2.0)
[ ] Article text files disk
[ ] Today Disk (circle the disk numbers below):
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

0.14 Disks:

[ ] Tutorial Disk
[ ] Auto Run Demo Disk
[ ] Paradise Disk
[ ] Best of COMAL
[ ] Bricks Tutorial
[ ] Utility Disk (circle which): 1 2
[ ] Slide Show disk (circle which): 1 2
[ ] Spanish COMAL
[ ] User Group 0.14 disks (circle numbers):
1 2 3 4 5 6 7 8 9 10 12

2.0 Disks:

[ ] Superchip Programs disk
[ ] Shareware (3 disk sides)
[ ] Read & Run
[ ] Math & Science
[ ] Typing disk
[ ] Cart Demo (circle which): 1 2 3 4
[ ] 2.0 User disks (circle choices): 11 13 14 15

§ = these disks assume you have the book

SYSTEMS

[ ] C64 COMAL 0.14 Starter Kit

includes Auto Run Demo, Tutorial, Paradise, Best of COMAL, and Sampler disks, *COMAL From A to Z*, *COMAL Workbook*, 12 issues of *COMAL Today*, and the 56 page *Index*... only $25.95. A great present for a friend! (combine this with Power Box on next page for a fantastic starting system)

[ ] Add $1 for C64 Keyboard Overlay

[ ] CP/M COMAL 2.10 Pak

Full COMAL system disk plus the DEMO disk, packed in a Doc Box with manual. Works in C128 CP/M mode. Only $45.90

[ ] Optional RUNTIME system ... only $3.95
[ ] Optional C128 Graphics Package $9.95

[ ] UniComal IBM PC COMAL 2.1*

$395.00 for the full fast system, with extensive tutorial and reference packed in a Doc Box. (save $200 off regular price)

[ ] $195.00 option (PLUS version). This adds a RUNTIME compiler and the Communication Package, with its own manuals packed in a second Doc Box (save another $100)

[ ] C64 2.0 Cartridge Deluxe Pak*

$125.90 - COMAL 2.0 Cartridge with Super Chip, Super Chip booklet, *Cartridge Graphics & Sound* book, *COMAL Handbook*, and all four Cartridge Demo disks.

[ ] C128 COMAL 2.0 Cart*

Special order price: $175.95. This cart works on the C128 in its native mode.

SCHOOLS

[ ] School COMAL Demo Pak

COMAL demo disks for IBM, CP/M, C64, and Apple. Plus a full set of *COMAL Today* issues 6-18 with 56 page *Index*. Only $10. (UPS USA shipping free, First Class add $9)

* = subject to overseas customs/shipping variations and availability.

more»

Fourth Private Sale - continued

BOOK SPECIALS

Choose any of the four Best Selling COMAL books below for just **$8.80** each. Plus, buy two and get one of the books for **free** (3 books for price of 2 ... plus your two free disks!)

[ ] **Introduction to COMAL 2.0**¤
#5 best seller by J William Leary
272 pages plus 64 page answer book
See page 31 for sample

[ ] **Beginning COMAL**¤
#8 best seller by Borge Christensen
333 pages - General Textbook
Written by the founder of COMAL
See page 32 for sample

[ ] **Foundations with COMAL**¤
#10 best seller by John Kelly
363 pages - General Textbook
See page 33 for sample

[ ] **COMAL Handbook**¤
#2 best seller by Len Lindsay
479 pages - Detailed Reference book
See page 34 for sample

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

Choose any of these three Best Selling books for just $1 each. Everyone should have a copy!

[ ] **COMAL From A to Z**¤
#1 best seller by Borge Christensen
64 pages - Mini 0.14 Reference book
Written by the founder of COMAL
See page 29 for sample

[ ] **COMAL Workbook**
#4 best seller by Gordon Shigley
69 pages - 0.14 Tutorial Workbook
See page 38 for sample

[ ] **COMAL Today - The Index**
#9 best seller by Kevin Quiggle
52 pages - Index to COMAL Today
See page 37 for sample

BRAND NEW

[ ] **Common COMAL Reference**
Detailed cross reference to the COMAL implementations in the USA by Len Lindsay (formerly called *COMAL Cross Reference*) $14.95 - See page 27 for sample

[ ] **CP/M COMAL Package Guide**
The guide to making your own packages for CP/M COMAL by Richard Bain - **$10.95** - See page 18 for sample

[ ] **POWER BOX**[db]
New!! Virtually all procedures and functions ever published in *COMAL Today* plus the complete *Library of Functions & Procedures* plus both Utility disk #1 and #2. **And, free during this sale: the Power Driver! $28.90**
[ ] Option ... add $5 for a Doc Box
[ ] Option ... add $1 for *COMAL From A To Z*

[ ] **PACKAGES COLLECTION**[db]
A complete package collection for the C64 Cartridge! Includes *COMAL 2.0 Packages*, *Package Library Vol 1* **and** *Package Library Vol 2*, plus Superchip On Disk with its source code! Over $100 in value, only **$29.90**
[ ] Option ... add $5 for a Doc Box
[ ] Option ... add $3 for *Cart Graphics/Sound*

[ ] **FONT PAK**
Font Disk #1 plus many more fonts released since then! Plus special font programs!
$12.95 ... three disks (one new this issue)

[ ] **SPRITE PAK**
Huge collection of sprite images, sprite editors, viewers, and other sprite programs!
$11.95 ... two disks (all new collection!)

[ ] **GRAPHICS PAK**
Picture heaven. Includes Slide Show system, Picture compactor system, Graphics Editor system, plus two disks full of compacted picture files! Five disks for only **$14.95**

db = Doc Box pages
¤ = while supply lasts (out of print)

more»

Fourth Private Sale - continued

MORE BOOKS

[] **Captain COMAL Gets Organized**[¤]
102 pages with disk by Len Lindsay
Modular programming applications tutorial
$8.95 - See page 35 for sample

[ ] **Library of Functions & Procedures**[db]
80 pages with disk by Kevin Quiggle
Over 100 procedures & functions for 0.14!
$8.95 - See page 26 for sample

[] **Cartridge Graphics & Sound**[¤]
#6 best seller by Captain COMAL
64 pages - 2.0 packages reference
$3.95 - See page 28 for sample

[ ] **COMAL 2.0 Packages**[db]
#7 best seller by Jesse Knight
108 pages with disk - package reference
$10.95 - See page 22 for sample

[ ] **Package Library Vol 1**[db]
compiled by David Stidolph
76 pages with disk - package collection
$10.95 - See page 20 for sample

[ ] **Package Library Vol 2**[db]
67 pages with disk - package collection
$10.95 - See page 21 for sample

[ ] **COMAL Collage**[db]
by Frank and Melody Tymon
168 pages with disk, 2.0 programming
$12.95 - See page 25 for sample

[ ] **3 Programs in Detail**[db]
82 pages with disk by Doug Bittinger
Three 2.0 application programs explained
$10.95 - See page 24 for sample

[ ] **Graph Paper**[db]
52 pages with disk by Garrett Hughes
Function graphing system for COMAL 2.0
$8.95 - See page 23 for sample

SUPER CHIP

Super Chip has become the standard for C64 COMAL 2.0. Now you can update your cartridge with over 100 commands too. Get the chip (black 2.0 cartridges only) or the disk loaded system (for black or beige cartridges). Both share the same commands (only the chip is available on power up, no disk load needed ... and also includes the Auto Start system).

[ ] **Super Chip - $9**
[ ] **Super Chip On Disk - $9**
[ ] **Both for only - $10.95**

Even More...

[ ] C64 2.0 Cart plain (free superchip) $85.95
[ ] European 12 Disk Set $23.95
[ ] Four Cart Demo disks $10.95
[ ] Eleven 0.14 User Group disks $23.95
[ ] Four 2.0 User Group disks $10.95
[ ] Source code to Super Chip On Disk $20.95
[ ] COMAL Yesterday $14.90
[ ] Free disk (our choice)
[ ] C64 Keyboard Overlay for 0.14 (folded) $1
[ ] Doc Box $5
[ ] Light Pen (McPen) (only one left) $19.95
[ ] Turtle's Source Book (list $24) $7 (one left)

[ ] **Guitar Disks**
New! Teaches guitar by playing songs while displaying the chords and words. Three disk set for COMAL 0.14, only **$9.95**

[ ] **C128 CP/M Graphics Package**
New! Adds over 20 turtle graphics commands including circle and fill for 640X200 monochrome graphics, **$9.95**

SPECIAL NOTES: We get <u>very</u> busy during our sale! Allow 3-4 weeks for delivery plus 2 weeks for checks to clear bank. Some disks may be supplied on the back side of another disk. ■

db = Doc Box pages
¤ = while supplies last (out of print)

Hi Lo Game

by Gary Franklin

Many COMAL programmers started programming in BASIC, and are intimidated by all the new structures found in COMAL. The more adventurous programmers start writing programs using every new trick they can find. Some quickly find procedures and **GOTO** disappears from their vocabulary. A structured programmer is born. For example, let's look at the classic Hi-Lo game - first in BASIC:

```
10 rem: this is a basic listing
20 n=int(rnd(0)*10)+1
30 input "guess a number from 1 to 10";g
40 if g>n then goto 90
50 if g<n then goto 110
60 print "correct"
70 input "do you want to play again";a$
80 if a$="y" then goto 20
85 end
90 print "too high"
100 goto 30
110 print "too low"
120 goto 30
```

In COMAL, this becomes:

```
DIM again$ of 1
play
//
PROC play
  number:=RND(1,10)
  guess'number
ENDPROC play
//
PROC guess'number
  INPUT "guess a number from 1 to 10 ":guess
  IF guess>number THEN
    PRINT "too high"
    guess'number
  ELIF guess<number THEN
    PRINT "too low"
    guess'number
  ELSE
    PRINT "correct"
    INPUT "do you want to play again? ":again$
    IF again$="y" THEN play
  ENDIF
  PRINT "recursion was here" //explained below
ENDPROC guess'number
```

The COMAL program listing is much more readable. Some might complain about it being longer, but that is the price you pay for clarity. A call to **guess'number** gets the point across much better than **GOTO 30**.

Beginning COMAL programmers might think this is a well written program. Once they learn the trick about how the **get'number** procedure can call itself over and over again until it finds the number it is looking for, they will want to use the trick in all their programs. A natural extension to the trick is to have the **get'number** procedure call the play procedure to start the second game once the first number is guessed and so. This trick is known in computer terms as *recursion.*

By now, you may realize what this article is leading to: programming advice. The above program is a classic example of a common, but serious, programming mistake. The problem is that you continually enter the **guess'number** procedure, but you never leave it (at least you don't leave before answering *no* to the *play again* question). If you really like the game, or if you are really bad at it, you might guess hundreds of numbers.

So, *what's the problem*, I hear you ask. The game works, but each time you call the **guess'number** procedure, COMAL needs to reserve a small piece of memory to tell it where to go once you get to **ENDPROC**. Normally, this doesn't matter. COMAL gives you a lot of memory and doesn't need very much for a procedure call. It frees up the memory once you leave the procedure so no one notices. However, if you call a procedure hundreds of times without ever leaving it, COMAL will eat

more»

up hundreds of small sections of memory requiring thousands of bytes. Sooner or later, you will get the dreaded **OUT OF MEMORY** error message. This is exactly what happens with the program above, although I expect you will get bored playing the game before you run out of memory. This is called infinite recursion.

A less obvious, but related problem is that **guess'number** calls **play** and **play** calls **guess'number**. This is called indirect recursion. Usually, indirect recursion is even more indirect. Perhaps a procedure **play** would call procedure **take'turn** which would call procedure **move'piece** which would call procedure **score**. Then **score** would start the process over again by calling **play**. The problem is the same, COMAL needs more and more memory to remember where to go after each procedure ends, but the procedures never end. Running out of memory is the only possible outcome short of nuclear intervention.

Now, let's explain that funny **PRINT** statement in the second to last line of the program. When you are playing the game, you will not see the message recursion was here. Does this mean you are not involved in recursion? No, it means recursion prevented COMAL from ever reaching the **PRINT** statement, hence you never see the message. This is what is meant above by procedures never end. The **guess'number** procedure never reaches the final **PRINT** statement. However, after you get tired of playing the game, you can see the message. In fact, after you say *no* to the *play again* prompt, COMAL will print recursion was here once for each guess you made starting from the first game. This should clearly show just how deeply nested you were in recursion.

If you have a COMAL cartridge, you might want to stop the game with the «*stop*» key. Then enter the **TRACE** command. I won't tell you what will happen, but as a hint, you may find the «*space bar*» helpful.

Ok, so recursion is a hard topic to master. Is there a better way? Yes, it is often better to avoid recursion completely. Here is another version of the Hi-Lo game which does not use recursion:

```
DIM again$ of 1
REPEAT
  play
  INPUT "do you want to play again? ":again$
UNTIL again$="n"
//
PROC play
  number=RND(1,10)
  REPEAT
    guess'number
  UNTIL guess=number
ENDPROC play
//
PROC guess'number
  INPUT "guess a number from 1 to 10 ":guess
  IF guess>number THEN
    PRINT "too high"
  ELIF guess<number THEN
    PRINT "too low"
  ELSE
    PRINT "correct"
  ENDIF
ENDPROC guess'number
```

Note: if you want to see explicitly how recursion eats up memory, add the following line to the start of both COMAL listings:

```
DIM waste$ of 9000 // the number varies
```

The length waste$ is dimensioned to must be altered for different COMAL versions. To find the number needed, first enter the program, then type the **SIZE** *command. Subtract 200 from the free memory and use that number in the* **DIM** *statement. The first COMAL program will quickly run out of memory but the second one can run forever.* ■

COMAL Coloring Book

by Dawn Hux

After two years of teaching BASIC and LOGO, I was looking for something new and exciting to challenge my programming students. When I heard about COMAL, I was immediately impressed with some features that made it attractive for teaching. The following may sound like a COMAL commercial but it points out the advantages that relate to a computer teacher in the classroom.

Current COMAL disks allow COMAL and the error messages to reside in the computer. Disk access is limited to loading and saving additional procedures or files. This is an important feature since my school uses one drive to serve four computers through a network and disk time has to be limited.

COMAL allows the directory to be displayed without overwriting programs already stored in memory; the downfall of many an unsaved BASIC program. It also can list programs as sequential files which can later be merged with other programs.

The language is structured and exposes my students to logic structures similar to those used in COBOL, FORTRAN, and PASCAL. The use of familiar keywords found in BASIC and LOGO makes COMAL easy to learn.

It has turtle graphics and an easier way to handle sprites.

The line editor helps in debugging and reducing syntax errors. The indention of logic structures aids in debugging logic errors.

The renumbering feature saves time for many students whose incomplete logic required inserting additional line numbers.

The 0.14 version can freely be copied which eliminates budget problems. Students can make a copy for use at home.

We started the year learning the language using the tutorial disk. Then we wrote several short programs and modified some of the programs found in *COMAL Today*. By mid-year we were ready to produce a more advanced project. Team effort is the norm in commercial program development so working as a team would be good practice for the students.

We evaluated a number of project possibilities and decided on a draw-and-paint program. This idea evolved into the *coloring'book* which made the project useful as well as entertaining. High school students wrote it to be used by elementary school children. The graphic capabilities of COMAL stimulated student interest and involvement better than responses to all-statistical programs. The COMAL procedure format allows a program to be divided into segments and assigned to different student teams. Unlike BASIC, the merging of procedures is not an ordeal.

The finished program appeared professional to the elementary coloring artist. This gave the high school programmer a "basketball star" status - a real academic ego booster.

Interaction with the young user was an eye opener and helped the high school students see the need for user friendly screens and good documentation. Student programmers often assume that the user will understand how to operate the program. Responding to bugs found by the users was a realistic follow up to this drill.

Logic is not limited to programming. One young student had trouble positioning the crayon accurately and often missed small targets. This resulted in filling the surrounding area, sometimes the entire screen. The next time he drew the picture he filled in the large areas

more»

around the trouble spots first. Then, if he misjudged the small spots, the already painted area just ignored the fill command and he could try again.

The ability to use published procedures was an advantage of COMAL not overlooked in this project. Some portions of this program were obtained from *COMAL Today* such as the joystick procedure on the *COMAL Today #4*, page 39 and a section of the *Bill'paint* program, *COMAL Today #14*, page 29.

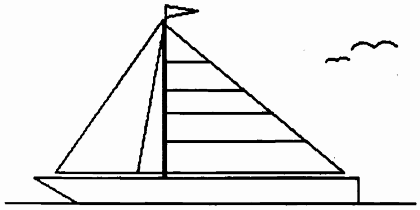
You may wish to speed up or slow down the crayon sprite movement. This is done in the **move** procedure. For young children, I selected a speed of 5 pixels per step because of their impatience. To obtain more accurate positioning in small areas you can change the speed to 3 or 2 pixels per step but it will take much longer to move across the page.

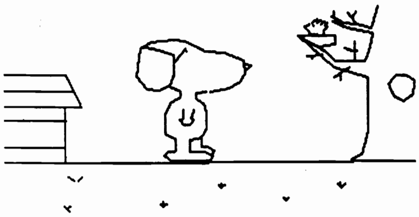
To write your own picture procedures, begin by deleting one of the existing picture procedures. The program on disk is now at its maximum size. Use COMAL turtle graphics reference material for instructions on commands available. There are two methods of instructions shown in our pictures. One is the **FORWARD 100, RIGHT 90** type of instructions used in the **sailboat** procedure. The other is the **MOVETO 120,90, DRAWTO 80,90** coordinate instructions used in the **town** and **cartoon** procedures. The **anyshape** procedure has been left in the program even though none of the art supplied requires it. Input the number of sides, length of sides, and angle to draw any polygon including circles.

I recommend you write the picture procedures separately and run them by adding **SETGRAPHIC 1** for multi-color mode and the name of your procedure on the first two lines before your picture procedure. You can then run the artwork to test the results of your commands. Once it is drawing correctly, delete the first two lines, renumber starting at 9000- (**RENUM 9000,1**) and **LIST** the procedure to disk.

Next **LOAD** the *coloring'book* and check to be sure there are no lines above 9000. Remember, you have to delete some of the student's pictures to make room and you may have to **RENUM**. Now **ENTER** your picture file from disk. Change the **menu** procedure to show the name of your new picture. Also change the **design** procedure which has the case statements to check on which picture has been selected. Once these two procedures have your picture's name, the program will recognize your new drawing. **RUN** the program and color it. If a bleed occurs at a corner of several lines, you can move the intersection over, up or down to get it to the edge of a screen block. This will eliminate the overwriting problem associated with too many colors assigned to one screen data block.

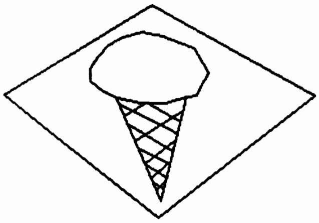
The names of the student artists whose work is incorporated into this version of the *coloring'book* are listed on the menu screen. Art was supplied by each Calvary Temple student participating in the COMAL classes but only seven could fit into memory. Three of the drawings were produced in another drawing program and converted to turtle graphics using a program devised entirely by the students. *Coloring'book* is on *Today Disk #19*.

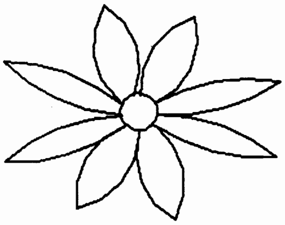


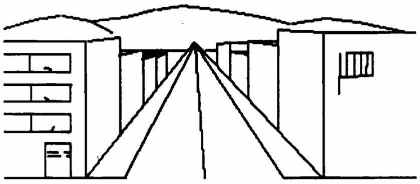


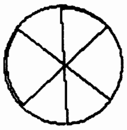
more»

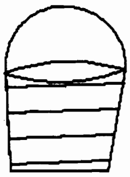
The Coloring Book - continued











```
print chr$(14+128),chr$(8)
dim image$ of 64
crayon
hidesprite 0
directions
repeat
 pages
until false
//
proc crayon
 for count:=1 to 63 do
  read d
  image$:=image$+chr$(d)
 endfor count
 image$:=image$+chr$(0)
 define 0,image$
 data 248,0,0,244,0,0,226,0,0,193,0,0
 data 128,128,0,64,64,0,32,32,0,16,16,0
 data 8,8,0,4,20,0,2,34,0,1,68,0,0,136,0,0,80,0,0,32,0
 data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
endproc crayon
//
proc directions
 settext
 border 6
 background 1
 pencolor 6
 print chr$(147)
 cursor(12,8)
 print "computerized coloring book"
 cursor(15,12)
 print "by the students of"
 cursor(18,5)
 print "calvary temple christian school"
 for count:=1 to 3500 do null
 print chr$(147)
 print " to use this coloring book program"
 print
 print " plug a joystick into port 2. use the "
 print
 print " joystick to move the crayon to the"
 print
 print " color you want and press the fire"
 print
 print " button.  move the crayon to the area"
 print
 print " you wish to color and press the fire"
 print
 print " button again."
 print
 print " to select another picture move the "
 print
 print " joystick to the 'x' and press the"
 print
 print " fire button."
 print
 print "      press space bar to continue"
 while key$=chr$(0) do null
endproc directions
//
proc pages
 repeat
  menu
 until choice>0 and choice<7
 setup
 spritepos 0,160,99
 design
```

more»

```
 done:=false
 repeat
  joystick(2,direction,button)
  move
  if button then
   if y<17 then
    color'change
   else
    fillin
   endif
  endif
 until done
endproc pages
//
proc menu
 settext
 pencolor 4
 print chr$(147)
 print " press the number of the picture you"
 print " would like to color. then press return."
 print
 print " 1 - sailboat by dawn hux"
 print
 print " 2 - cartoon by matt hartkop"
 print
 print " 3 - beach by kevin beachy"
 print
 print " 4 - ice cream cone by berin loritsch"
 print
 print " 5 - flower by christyn reid"
 print
 print " 6 - town by jeff heglund"
 print
 input choice
endproc menu
//
proc setup
 increment:=5
 x:=160
 y:=99
 setgraphic 1
 hideturtle
 pencolor 2
 colorbar
 image$:=""
 identify 0,
 spritecolor 0,2
 spritesize 0,1,1
 spritepos 0,160,99
endproc setup
//
proc colorbar
 border 0
 background 1
 pencolor 0
 setgraphic 1
 hideturtle
 color:=2
 moveto 0,17
 drawto 319,17
 for z:=39 to 279 step 16 do
  moveto z,0
  drawto z,17
 endfor z
 color:=0
 fill 42,0
 color:=2
 for j:=55 to 263 step 16 do
  pencolor color
  fill j+5,0
  color:=color+1
 endfor j
 pencolor 0
 moveto 2,2
 drawto 14,14
 moveto 2,14
 drawto 14,2
endproc colorbar
//
proc design
 case choice of //****menu****
 when 1
  sailboat
 when 2
  cartoon
 when 3
  beach
 when 4
  cone
 when 5
  flower
 when 6
```

more»

The Coloring Book - continued

```
  town
 otherwise
 endcase
endproc design
//
proc joystick(p,ref d,ref f)
closed // wrap line
 if p=1 then
  m:=peek(56321)
 elif p=2 then
  m:=peek(56320)
 else
  return
 endif
 f:=1-((m mod 32) div 16)
 case 15-(m mod 16) of
 when 1
  d:=1
 when 2
  d:=5
 when 4
  d:=7
 when 5
  d:=8
 when 6
  d:=6
 when 8
  d:=3
 when 9
  d:=2
 when 10
  d:=4
 otherwise
  d:=0
 endcase
endproc joystick
//
proc cursor(row,col) closed
 addr:=1024+(row-1)*40
 poke 209,addr mod 256
 poke 210,addr div 256
 poke 211,col-1
 poke 214,row-1
endproc cursor
//
proc move
```

```
 case direction of
 when 1
  y:+increment
 when 2
  y:+increment; x:+increment
 when 3
  x:+increment
 when 4
  x:+increment; y:-increment
 when 5
  y:-increment
 when 6
  x:-increment; y:-increment
 when 7
  x:-increment
 when 8
  x:-increment; y:+increment
 otherwise
  null
 endcase
 if x<0 then x:=0
 if y<0 then y:=0
 if x>319 then x:=319
 if y>199 then y:=199
 spritepos 0,x,y
endproc move
//
proc color'change
 if x>=39 then
  color:=(x-23) div 16
  if color=1 then color:=0
  if color>15 then color:=15
  pencolor color
  border color
 else
  done:=true
 endif
endproc color'change
//
proc fillin
 fill x,y
endproc fillin
//
proc sailboat
 pencolor 0
 pendown
```

```
moveto 277,39
//boat hull******
left 90
forward 220
right 30
forward 39
right 150
forward 103
left 90
forward 120
//main sail*****
right 130
forward 180
right 140
forward 140
//front sail****
right 90
forward 120
left 145
forward 146
left 125
forward 84
//return to hull**
right 90
forward 4
left 90
forward 151
right 90
forward 20
left 90
forward 45
right 180
forward 322 //water
//sail design****
moveto 125,88
setheading 90
forward 110
moveto 125,110
forward 84
moveto 125,128
forward 61
moveto 125,150
forward 36
//draw in sky****
moveto 270,160
setheading 40
```

more»

```
  for count:=1 to 10 do
    forward 2
    right 10
  endfor count
  setheading 40
  for count:=1 to 10 do
    forward 2
    right 10
  endfor count
  moveto 250,150
  setheading 40
  for count:=1 to 10 do
    forward 1
    right 10
  endfor count
  setheading 40
  for count:=1 to 10 do
    forward 1
    right 10
  endfor count
  moveto 125,180
  setheading 190
  forward 119
  //flag*****
  moveto 126,179
  setheading 0
  forward 15
  right 100
  forward 25
  right 160
  forward 25
endproc sailboat
//
proc beach
  moveto 170,60
  right 90
  forward 75
  left 80
  forward 80
  left 120
  for x:=1 to 19 do
    forward 5
    right 2
  endfor x
  right 142
  for x:=1 to 19 do
    forward 5
    right 2
  endfor x
  moveto 170,60
  left 110
  forward 75
  back 10
  right 90
  forward 90
  back 90
  right 90
  forward 20
  left 90
  forward 85
  back 85
  right 90
  forward 20
  left 90
  forward 80
  back 80
  right 90
  forward 20
  left 90
  forward 75
  back 75
  right 90
  forward 5
  left 180
  forward 80
  for x:=1 to 75 do
    forward 2
    right 2.5
  endfor x
  moveto 100,60
  setheading 270
  for x:=1 to 143 do
    forward 2
    right 2.5
  endfor x
  right 90
  forward 90
  right 90
  for x:=1 to 20 do
    forward 2
    right 2.5
  endfor x
  right 90
  forward 90
  right 90
  for x:=1 to 32 do
    forward 2
    right 2.5
  endfor x
  right 90
  forward 90
endproc beach
//
proc anyshape(sides,
distance,angle)//wrap
  for count:=1 to sides
    forward distance
    right angle
  endfor count
endproc anyshape
//
proc cone
  moveto 123,118
  drawto 141,114
  drawto 160,114
  drawto 176,119
  drawto 156,38
  drawto 122,118
  drawto 109,125
  drawto 102,131
  drawto 102,141
  drawto 108,150
  drawto 115,158
  drawto 125,164
  drawto 142,169
  drawto 158,166
  drawto 182,157
  drawto 189,145
  drawto 193,137
  drawto 189,130
  drawto 184,120
  drawto 176,119
  moveto 125,113
  drawto 168,91
  moveto 132,97
  drawto 165,77
  moveto 139,79
  drawto 163,67
  moveto 145,66
  drawto 161,56
  moveto 150,53
  drawto 157,48
  moveto 153,47
  drawto 160,54
  moveto 149,56
  drawto 161,67
  moveto 144,67
  drawto 167,85
  moveto 137,85
  drawto 173,108
  moveto 128,104
  drawto 143,113
  moveto 148,113
  drawto 171,100
  moveto 265,83
  moveto 154,25
  drawto 35,120
  drawto 150,190
  drawto 275,120
  drawto 154,25
endproc cone
//
proc flower
  moveto 140,88
  setheading 270
  for x:=1 to 36 do
    forward 2.5
    right 10
  endfor x
  moveto 140,115
  setheading 5
  petal2
  setheading 65
  petal3
  setheading 95
  petal3
  setheading 138
  forward 2
  petal2
  setheading 190
  forward 3
  petal2
  setheading 238
  petal3
```

more»

```
 setheading 280
 forward 3
 petal3
 setheading 322
 forward 5
 petal2
endproc flower
//
proc petal2
 for count:=1 to 4 do
  forward 20
  right 20
 endfor count
 right 90
 for count:=1 to 4 do
  forward 20
  right 20
 endfor count
endproc petal2
//
proc petal3
 for count:=1 to 4 do
  forward 25
  right 10
 endfor count
 right 135
 for count:=1 to 4 do
  forward 25
  right 10
 endfor count
endproc petal3
//
proc town
 moveto 149,142
 drawto 246,37
 moveto 148,142
 drawto 64,36
 moveto 94,36
 drawto 149,141
 drawto 217,36
 drawto 319,36
 moveto 95,36
 drawto 0,36
 moveto 64,36
 drawto 64,143
 drawto 0,143
 moveto 248,36
 drawto 248,150
 moveto 319,150
 drawto 249,150
 moveto 249,150
 drawto 213,152
 drawto 213,73
 moveto 66,143
 drawto 91,146
 moveto 91,69
 drawto 91,146
 moveto 90,132
 drawto 107,136
 moveto 109,93
 drawto 109,136
 moveto 212,140
 drawto 190,143
 moveto 190,99
 drawto 190,143
 moveto 190,133
 drawto 176,135
 drawto 176,113
 moveto 176,134
 drawto 176,142
 drawto 166,141
 drawto 166,124
 moveto 108,129
 drawto 119,132
 moveto 119,106
 drawto 119,132
 moveto 119,130
 drawto 127,134
 drawto 127,116
 moveto 156,36
 drawto 149,141
 moveto 178,135
 drawto 190,135
 moveto 191,143
 drawto 212,143
 moveto 109,136
 drawto 91,136
 moveto 118,132
 drawto 109,132
 moveto 150,135
 drawto 150,138
 moveto 157,136
 drawto 166,136
 moveto 144,136
 drawto 109,136
 moveto 127,134
 drawto 110,134
 moveto 0,153
 drawto 25,160
 drawto 44,163
 moveto 46,163
 drawto 68,159
 moveto 70,159
 drawto 96,162
 drawto 118,168
 moveto 120,168
 drawto 140,171
 drawto 173,167
 drawto 202,162
 drawto 222,157
 moveto 224,157
 drawto 250,161
 moveto 251,161
 drawto 275,158
 drawto 313,150
 moveto 68,159
 drawto 85,150
 moveto 52,133
 drawto 52,119
 moveto 0,119
 drawto 52,119
 moveto 52,133
 drawto 0,133
 moveto 20,133
 drawto 20,119
 moveto 52,109
 drawto 52,96
 drawto 0,96
 moveto 53,109
 drawto 0,109
 moveto 21,109
 drawto 21,96
 moveto 52,86
 drawto 52,72
 drawto 0,72
 moveto 52,86
 drawto 0,86
 moveto 21,86
 drawto 21,72
 moveto 52,63
 drawto 52,36
 moveto 52,63
 drawto 32,63
 moveto 32,36
 drawto 32,63
 moveto 36,72
 drawto 39,76
 moveto 36,96
 drawto 32,99
 moveto 36,119
 drawto 36,121
 moveto 37,121
 drawto 32,122
 moveto 34,56
 drawto 42,56
 moveto 34,53
 drawto 44,53
 moveto 52,56
 drawto 47,56
 moveto 52,53
 drawto 49,53
 moveto 266,134
 drawto 286,134
 drawto 286,116
 drawto 261,115
 drawto 261,133
 moveto 266,134
 drawto 266,116
 moveto 272,116
 drawto 272,133
 moveto 277,133
 drawto 277,116
 moveto 281,116
 drawto 282,133
 moveto 261,128
 drawto 260,102
 moveto 177,142
 drawto 190,142
endproc town
//
proc cartoon
 moveto 161,132
 drawto 176,135
 drawto 184,140
```

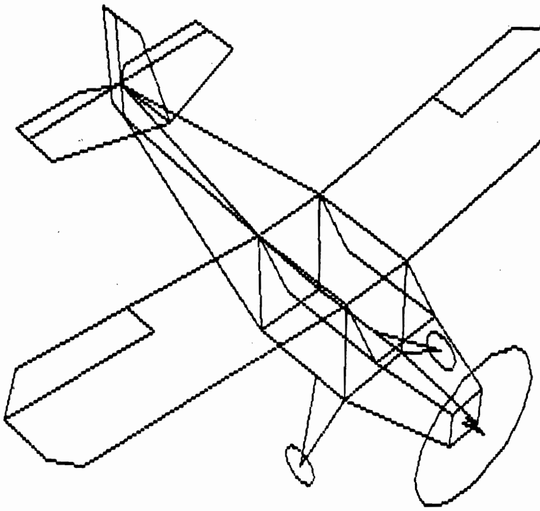
more»

The Coloring Book - continued

```
drawto 188,147
drawto 187,157
drawto 172,163
drawto 157,164
drawto 141,172
drawto 132,173
drawto 115,171
drawto 107,166
drawto 104,144
drawto 112,134
drawto 122,133
drawto 130,137
drawto 133,146
moveto 133,155
drawto 142,166
moveto 142,165
drawto 143,164
moveto 187,153
drawto 190,154
drawto 193,153
drawto 188,149
moveto 130,136
drawto 137,133
moveto 158,132
drawto 150,131
drawto 150,127
moveto 138,132
drawto 138,128
drawto 131,124
drawto 126,114
drawto 126,99
drawto 134,95
drawto 134,86
moveto 151,126
drawto 156,123
drawto 158,115
drawto 158,103
drawto 154,96
drawto 145,94
drawto 145,85
drawto 157,85
drawto 163,82
drawto 161,78
drawto 130,78
drawto 125,82
drawto 128,85
drawto 133,85
moveto 154,95
drawto 154,89
drawto 159,89
drawto 164,86
drawto 162,82
moveto 140,118
drawto 140,111
drawto 137,110
drawto 140,106
drawto 145,106
drawto 148,110
drawto 146,111
drawto 148,118
moveto 133,144
drawto 133,161
drawto 121,165
drawto 110,167
moveto 270,79
drawto 278,83
drawto 283,101
drawto 283,143
drawto 257,151
drawto 229,176
drawto 251,163
drawto 258,156
drawto 281,156
drawto 285,167
drawto 287,178
drawto 267,182
drawto 256,187
drawto 275,185
drawto 286,182
drawto 291,199
moveto 273,157
drawto 273,171
moveto 271,164
drawto 265,166
moveto 273,163
drawto 275,164
moveto 269,151
drawto 257,143
moveto 262,146
drawto 262,141
moveto 248,161
drawto 236,160
moveto 240,160
drawto 240,158
moveto 233,173
drawto 246,167
drawto 258,169
drawto 258,176
drawto 230,176
moveto 238,185
drawto 234,183
drawto 236,181
drawto 238,178
moveto 238,177
drawto 244,177
moveto 240,186
drawto 244,189
moveto 244,186
drawto 247,188
moveto 247,185
drawto 250,187
moveto 250,184
drawto 252,186
moveto 252,183
drawto 249,180
drawto 249,177
moveto 241,184
drawto 241,184
moveto 263,187
drawto 255,191
moveto 257,191
drawto 257,195
moveto 274,186
drawto 268,194
moveto 271,190
drawto 274,195
moveto 305,145
drawto 299,138
drawto 303,129
drawto 311,125
drawto 318,130
drawto 319,141
drawto 313,145
drawto 306,145
moveto 270,78
drawto 319,78
moveto 48,77
drawto 48,120
drawto 0,120
moveto 0,76
drawto 48,76
moveto 48,120
drawto 61,120
drawto 49,145
drawto 0,145
moveto 0,132
drawto 54,132
moveto 49,109
drawto 0,109
moveto 0,88
drawto 47,88
moveto 49,76
drawto 271,77
moveto 259,60
drawto 262,57
drawto 265,60
moveto 261,59
drawto 261,61
moveto 217,49
moveto 217,49
drawto 219,47
drawto 221,49
moveto 219,48
drawto 218,49
moveto 167,59
drawto 169,57
drawto 172,59
moveto 169,57
drawto 169,60
moveto 124,42
drawto 122,44
moveto 124,42
drawto 127,44
moveto 124,42
drawto 124,46
moveto 51,38
drawto 47,42
moveto 49,40
drawto 52,42
moveto 58,62
drawto 61,65
moveto 55,62
drawto 50,65
endproc cartoon ■
```

Rotating 3D Image

by Luther Hux



THREE-QUARTER VIEW

Demo/rotate'3d and *rotate'3d'plane* on *Today Disk #19* both rotate a 3D perspective drawing around three axis. At first glance this may appear to be a repeat of the 3D projection article presented in *COMAL Today #13* but a look inside the program shows a different and more compact approach with data line input so you can easily change the image.

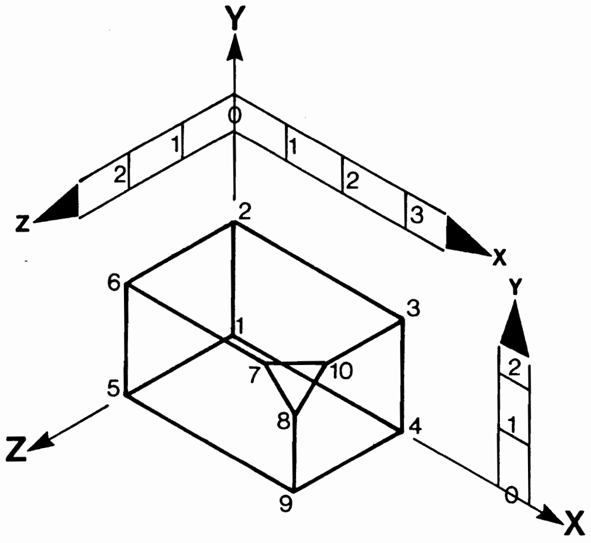
The rotation routine is the same in each program. It is reasonably easy to comprehend in theory even if you can't follow it to the last variable.

The algorithm was presented by Assoc. Prof C. Smith at a college course I attended on how CAD programs are designed. My wife, Dawn, assisted with converting my class programs to COMAL and wrote the **drawline** procedure to emulate the IBM BASIC keyword LINE. My contribution to the program is the airplane data. I design and draw model airplanes so I naturally chose an airplane for this project.

Sample input data is included in remark statements but I'll repeat it here to get you started. Answer scale with 15. Use a larger number for a larger image. For yaw, roll, pitch type in 25, 35, 10 for degrees of rotation on each axis respectively. The unrotated model (0, 0, 0) is a side view flying to the right. For distance to eye and to projection type in 60, 60 for a normal view or 40, 60 for a closer view. Once the drawing is complete, type «*f1*» to get back to the text screen. To rerun the program type a 1 or any other number to quit. Now try any numbers that suit your fancy and watch the outcome. The following inputs for scale, rotation, and projection create an unusual perspective: 5 / 0, 25, 0 / 15, 50.

Drawing Your Own Shapes

I'm sure you will have a favorite item other than an airplane you may want to draw. Here's an introduction to writing your own data lines.



DEMONSTRATION RECTANGLE

The format for the data is X, Y, Z, P. For ease of editing keep each line separate and label sections of the drawing as seen in the airplane data. The X axis is horizontal and it progresses

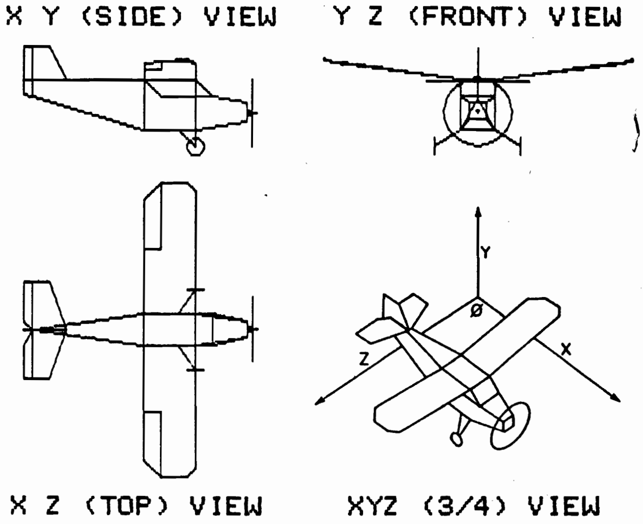
more»

from 0, from left to right. The Y axis is vertical and it progresses from 0, from bottom up. The Z axis is horizontal at right angles to the X an Y axis and on our illustration it progresses from 0, from right to left. On the monitor the Z axis points straight at you.

Numbering in all axis starts at zero at the intersection. Shapes that go behind the intersection are represented in negative numbers. The P variable represents pen up or pen down. Any choice or numbers will work but since CAD plotters at the college use 3 for pen up and 2 for pen down I kept that system. In the drawline procedure, the input of 3 receives an **MOVETO** response and the input of 2 receives a **DRAWTO** response.

The corners of the demonstration rectangle are numbered in a path which eliminates a lot of pen-ups to complete the drawing. The rectangle is 3 units long (note the scale), two units tall and two units deep. The coordinates for corner #1 is 0,0,0,3. The zeros sets the pen at the corner of all three axis and the 3 holds the "pen up" so it will not draw a line on the way to the corner. Corner #2 is identified as 0,2,0,2. There is 0 movement on the X axis, move 2 units up the Y axis and 0 movement on the Z axis and the last 2 puts the pen down so the line will be drawn. Corner #3 is identified as 3,2,0,2. And so on. Complete the coordinates in the numbered sequence through #9. Then pick up the pen and move to draw the required cross bars. Complete the triangle of the chipped corner which acts as a key to which way the shape has rotated. Count the number of data lines and add a data line at the beginning with that number.

Once you understand how to write and rotate the rectangle begin on your own projects. Sketch a three view (or three-quarter view) of your image on graph paper. Make the original drawing between 3 to 20 units long. Too small or large an original will complicate the math. Then choose the path of each line and begin writing your data. Remember to make the first data line the number of lines of data you have written which tells the **FOR ENDFOR** loop how many lines of data to read in the variable N. Do not include this line in your count of data lines.

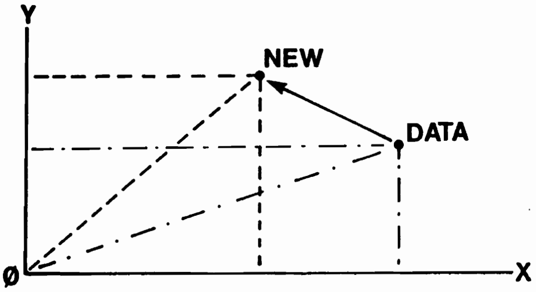


You'll notice that half of my airplane data is negative numbers to two decimal places. Producing that data without some type of detailed guide would have been very difficult. The original data, identical in appearance to the present model, was drawn as all positive numbers. But there was a problem when I ran the program with the rotation point at the corner. The model would swing off the screen and out of view when wide angles of rotation were entered. I decided to move the rotation point to the center of the model so it could rotate within the screen.

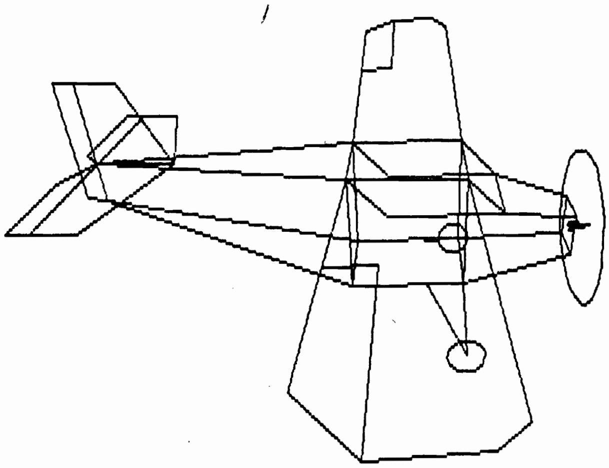
I then wrote a program that would move each axis to center and rewrite new data. A copy of the original program was stripped of everything except data lines. This was listed to the disk as a sequential file. A program was written that

more»

would read this file, subtract the appropriate amount from each data/axis point and write a new file with complete data lines. Hence the awesome looking complexity of the data was originally done in the comfort of positive numbers and converted in about a minute.



CALCULATING POINT MOVEMENT



FISH-EYE PERSPECTIVE

The Algorithm

In the rotation routine the sin'x,sin'y,sin'z and cos'x,cos'y,cos'z provides the radian of each X,Y Z input's sin and cos. The data line specifies the position of a point in the model with no rotation adjustments. Using sin and cos calculations the algorithm plots X and Y adjustments based on the rotation inputs. It then plots what effect the third dimension would have and makes additional adjustments along the X and Y axis to create the illusion of 3D. It is important to note there is no true 3D image in the computers memory. In fact, there is also no airplane in memory either. It is plotted one single point at a time based on the algorithm and the processor does not have the last point in memory and has not yet read the next point. Only the video output memory has the overall picture. The program and the screen are only 2D. The illusion of 3D is created in exactly the same fashion as an artist sketching an airplane on a sheet of paper. The artist adjusts the positions of each point in the drawing to create the 3D appearance.

The perspective routine provides the degree of distortion required for proper perspective appearance. The value of to'eye establishes the distance from the screen to the eye. Keep in mind that this input has nothing to do with the actual distance from the monitor screen to your eye. A small number will increase the distortion as if you were very close to a 3D image. The value of to'proj establishes the distance from the screen to the point of projection.

Confused? Try this illustration. Imagine a projector behind a rear projection screen. You are standing in front of the screen looking at the image projected from behind. The further you move the projector from the screen the larger the image. Then imagine moving your eye (to'eye) up close to the screen. The closer you get the more distorted the image would be until you get so close you get a fish-eye lens effect. Of course, a very close view would also produce a very large image so you must move the projector (to'proj) closer to the screen to get the image back within the screen. It is the ratio between the two (and the scale factor)

more»

that produces the desired distortion and size. I suggest you keep the numbers between 10 and 100 to prevent an overflow or taking a long time to calculate. Note: you cannot go inside the model.

Three additional modifications are made to the 2D data forwarded from the perspective routine. First, units are added to the X and Y output to center the model on the screen. The 0, 0 start position of your screen is in the lower left corner. Feel free to change this number if your object isn't positioned to suit you. Second, the output is multiplied by the scale factor scale to determine the final size of the model. Third, after the X, Y values are sent to the drawline procedure, the Y value is divided by 1.28 to square the model on the screen. Without this alteration the model would be distorted vertically. To test your screen simply run the program with a large scale of about 20 and the X,Y,Z value of 90,0,0. This will turn the model straight toward you so you can measure the roundness of the propeller. If it proves to be out of round, first check to see if you can adjust your screens height as you can on a Magnavox monitor. If not, alter the 1.28 crunch number as needed.

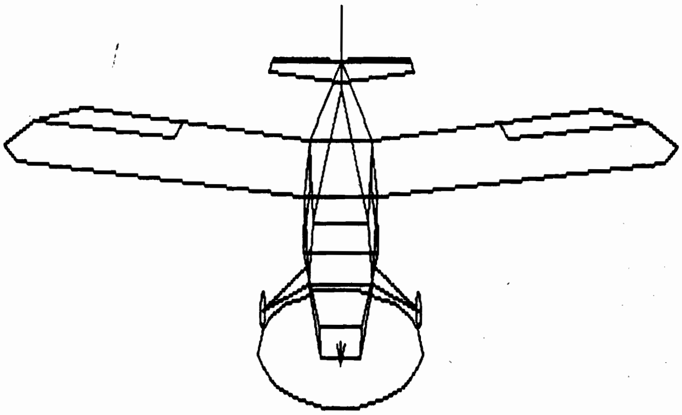
One problem with a wire drawing is interpreting which direction the model is actually facing. You may decide you are looking at the top of the model but at a later look decide it's the bottom. Remembering that the left wing is drawn first may help you decide. Also, the addition of perspective helps to introduce the correct view. If the model looks terribly awkward then you may be fixed on the wrong point of view.

The X, Y output can be sent to a plotter proc instead of the drawing proc. Since plotters do not require the Y/1.28 adjustment, it was placed in the drawing proc instead of with the X, Y output in the algorithm.

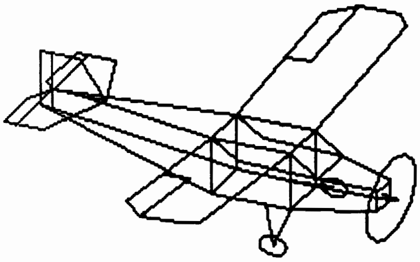
If you have video equipment you can output the screen to your video tape and record a few frames of individual screens of progressing angles of view. The finished video will then give the appearance of animation.

An extra bonus is hidden in the program. If you wish to add X, Y, Z lines and labels to the drawing for clear identification of the axis, like the lines in the 3/4 view, simply add 20 to the first data line to read a total of 194 data lines. This increases the read loop to include the axis line data.

Have a nice flight.



FISH-EYE PERSPECTIVE



press f1 to continue

more»

*** 3d Rectangle ***

```
//orthographic projection with perspective
//program & artwork by luther hux
background 6
border 6
draw'again:=1
while draw'again=1 do
 print "rotate an airplane"
 input "scale ": scale
 input "rotation - yaw, roll, pitch ":
 yaw'x,roll'y,pitch'z // wrap line
 input "distance to eye, to projector ":
 to'eye,to'proj // wrap line
 setgraphic 0
 hideturtle
 //data required = 20
 //**** rotation routine
 sin'x:=sin(yaw'x*3.1416/180)
 cos'x:=cos(yaw'x*3.1416/180)
 sin'y:=sin(roll'y*3.1416/180)
 cos'y:=cos(roll'y*3.1416/180)
 sin'z:=sin(pitch'z*3.1416/180)
 cos'z:=cos(pitch'z*3.1416/180)
 restore
 read line'count
 for loops:=1 to line'count do
  read axis'x2,axis'y2,axis'z2,pen
  axis'x3:=axis'x2*cos'x-axis'z2*sin'x
  axis'z3:=axis'x2*sin'x+axis'z2*cos'x
  axis'z4:=axis'z3*cos'y+axis'y2*sin'y
  axis'y4:=-axis'z3*sin'y+axis'y2*cos'y
  axis'x4:=axis'x3*cos'z+axis'y4*sin'z
  axis'y5:=-axis'x3*sin'z+axis'y4*cos'z
  //**** perspective adjustment
  axis'x6:=axis'x4+(to'eye-axis'z4-to'proj
  )*(-axis'x4)/(to'eye-axis'z4) // wrap line
  axis'y6:=axis'y5+(-axis'y5)*(to'eye-axis'z4-
  to'proj)/(to'eye-axis'z4) // wrap line
  //**** x, y output
  center'x:=170
  center'y:=120
  x:=center'x+axis'x6*scale
  y:=center'y+axis'y6*scale
  drawline
 endfor loops
 input "draw again  yes=1  ": draw'again
endwhile
//**** drawing procedure
proc drawline
 if pen=3 then
  penup
  moveto x,y/1.28
 else
  pendown
  drawto x,y/1.28
 endif
endproc drawline
print "happy landing"
//
data 20
data 0,0,0,3
data 0,2,0,2
data 3,2,0,2
data 3,0,0,2
data 0,0,0,2
data 0,0,2,2
data 0,2,2,2
data 2.5,2,2,2
data 3,1.5,2,2
data 3,0,2,2
data 0,0,2,2
data 3,1.5,2,3
data 3,2,1.5,2
data 2.5,2,2,2
data 3,2,1.5,3
data 3,2,0,2
data 0,2,0,3
data 0,2,2,2
data 3,0,0,3
data 3,0,2,2
```

*** 3d Airplane ***

```
//orthographic projection with perspective
//comal .14 version
//comal program & artwork
// by luther & dawn hux
//class project from cad design class
// given by no. va. comm. college
//7620 trammell road
//annandale, va 22003
```

more»

```
//(703) 573-1244
//
//sample scale = 15
//sample rotation = 25,35,10
//sample perspective distance to eye & to
// projector = 60,60  (or 40,60)
//note that the unrotated model is a side view
// flying to the right.
background 6
border 6
pencolor 1
draw'again:=1
while draw'again=1 do
 print "rotate an airplane"
 input "scale ": scale
 input "rotation - yaw, roll, pitch ":
 yaw'x,roll'y,pitch'z // wrap line
 input "distance to eye, to projector ":
 to'eye,to'proj // wrap line
 setgraphic 0
 hideturtle
 //data required = 174 for airplane
 // or 194 to add x,y,z axis lines.
 //**** rotation routine
 sin'x:=sin(yaw'x*3.1416/180)
 cos'x:=cos(yaw'x*3.1416/180)
 sin'y:=sin(roll'y*3.1416/180)
 cos'y:=cos(roll'y*3.1416/180)
 sin'z:=sin(pitch'z*3.1416/180)
 cos'z:=cos(pitch'z*3.1416/180)
 restore
 read line'count
 for loops:=1 to line'count do
  read axis'x2,axis'y2,axis'z2,pen
  axis'x3:=axis'x2*cos'x-axis'z2*sin'x
  axis'z3:=axis'x2*sin'x+axis'z2*cos'x
  axis'z4:=axis'z3*cos'y+axis'y2*sin'y
  axis'y4:=-axis'z3*sin'y+axis'y2*cos'y
  axis'x4:=axis'x3*cos'z+axis'y4*sin'z
  axis'y5:=-axis'x3*sin'z+axis'y4*cos'z
  //**** perspective adjustment
  axis'x6:=axis'x4+(to'eye-axis'z4-to'proj
  )*(-axis'x4)/(to'eye-axis'z4) // wrap line
  axis'y6:=axis'y5+(-axis'y5)*(to'eye-axis'z4-
  to'proj)/(to'eye-axis'z4) // wrap line
  //**** x, y output
  center'x:=170
  center'y:=120
  x:=center'x+axis'x6*scale
  y:=center'y+axis'y6*scale
  drawline
 endfor loops
 input "draw again  yes=1  ": draw'again
endwhile
//**** drawing procedure
proc drawline
 if pen=3 then
  penup
  moveto x,y/1.28
 else
  pendown
  drawto x,y/1.28
 endif
endproc drawline
print "happy landing"
//
data 174 //194 to add axis lines
//*** rudder outline
data -9,0,0,3
data -9,3,0,2
data -7.5,3,0,2
data -6.5,1,0,2
data -8.5,1,0,2
//*** left fuselage outline
data -8.5,1,0,3
data -2,1,-1,2
data 1,1,-1,2
data 2,0,-1,2
data 4,-.41,-.5,2
data 4,-1.5,-.5,2
data 1,-2,-1,2
data -2,-2,-1,2
data -9,0,0,2
//*** right fuselage outline
data -2,-2,-1,3
data -2,1,-1,2
data -1,0,-1,2
data 2,0,-1,2
data -7.5,1,0,3
data -2,1,1,2
data 1,1,1,2
data 2,0,1,2
```

more»

```
data 4,-.41,.5,2
data 4,-1.5,.5,2
data 1,-2,1,2
data -2,-2,1,2
data -9,0,0,2
//*** cross bars
data 1,1,1,3
data 1,-2,1,2
data 1,-2,-1,2
data 1,1,-1,2
data -2,-2,1,3
data -2,1,1,2
data -1,0,1,2
data 2,0,1,2
data 4,-.5,-.5,3
data 4,-.5,.5,2
data -2,-2,-1,3
data -2,-2,1,2
data 4,-1.5,-.5,3
data 4,-1.5,.5,2
data 2,0,-1,3
data 2,0,1,2
//*** wing outline
data 1,1,-1,3
data 1,2,-8.5,2
data .5,2.19,-9,2
data -1,2.09,-9,2
data -2,2,-8,2
data -2,1,-1,2
data -2,1,1,2
data -2,2,8,2
data -1,2.09,9,2
data .5,2.19,9,2
data 1,2,8.5,2
data 1,1,1,2
data 1,1,-1,2
//*** ailerons
data -1,2.09,-9,3
data -1,1.59,-5,2
data -2,1.59,-5,2
data -1,2.09,9,3
data -1,1.59,5,2
data -2,1.59,5,2
//*** stab outline
data -8.5,3,0,3
data -8.5,-.11,0,2
data -6.5,1,0,3
data -7.5,1,-3,2
data -9,1,-3,2
data -9,1,-.5,2
data -8.5,1,0,2
data -9,1,1,2
data -9,1,3,2
data -7.5,1,3,2
data -6.5,1,0,2
data -8.5,1,-3,3
data -8.5,1,3,2
//*** left wheel
data 1,-2.5,2.5,3
data 1.09,-2.51,2.5,2
data 1.19,-2.54,2.5,2
data 1.29,-2.6,2.5,2
data 1.39,-2.7,2.5,2
data 1.44,-2.8,2.5,2
data 1.48,-2.9,2.5,2
data 1.5,-3,2.5,2
data 1.48,-3.1,2.5,2
data 1.44,-3.2,2.5,2
data 1.39,-3.3,2.5,2
data 1.29,-3.4,2.5,2
data 1.19,-3.45,2.5,2
data 1.09,-3.48,2.5,2
data 1,-3.5,2.5,2
data .89,-3.48,2.5,2
data .79,-3.45,2.5,2
data .69,-3.4,2.5,2
data .59,-3.3,2.5,2
data .54,-3.2,2.5,2
data .51,-3.1,2.5,2
data .5,-3,2.5,2
data .51,-2.9,2.5,2
data .54,-2.8,2.5,2
data .59,-2.7,2.5,2
data .69,-2.6,2.5,2
data .79,-2.55,2.5,2
data .89,-2.52,2.5,2
data 1,-2.5,2.5,2
//*** right wheel
data 1,-2.5,-2.5,3
data 1.09,-2.52,-2.5,2
data 1.19,-2.55,-2.5,2
data 1.29,-2.6,-2.5,2
data 1.39,-2.7,-2.5,2
data 1.44,-2.8,-2.5,2
data 1.48,-2.9,-2.5,2
data 1.5,-3,-2.5,2
data 1.48,-3.1,-2.5,2
data 1.44,-3.2,-2.5,2
data 1.39,-3.3,-2.5,2
data 1.29,-3.4,-2.5,2
data 1.19,-3.45,-2.5,2
data 1.09,-3.48,-2.5,2
data 1,-3.5,-2.5,2
data .89,-3.48,-2.5,2
data .79,-3.45,-2.5,2
data .69,-3.4,-2.5,2
data .59,-3.3,-2.5,2
data .54,-3.2,-2.5,2
data .51,-3.1,-2.5,2
data .5,-3,-2.5,2
data .51,-2.9,-2.5,2
data .54,-2.8,-2.5,2
data .59,-2.7,-2.5,2
data .69,-2.6,-2.5,2
data .79,-2.55,-2.5,2
data .89,-2.52,-2.5,2
data 1,-2.5,-2.5,2
//*** landing gear
data 1,-2,-1,3
data 1,-3,-2.5,2
data 0,-2,-1,2
data 0,-2,1,3
data 1,-3,2.5,2
data 1,-2,1,2
//*** propeller
data 4.3,1,0,3
data 4.3,.95,-.4,2
data 4.3,.8,-.8,2
data 4.3,.6,-1.2,2
data 4.3,.2,-1.6,2
data 4.3,-.2,-1.8,2
data 4.3,-.6,-1.95,2
data 4.3,-1,-2,2
data 4.3,-1.4,-1.95,2
data 4.3,-1.8,-1.8,2
data 4.3,-2.2,-1.6,2
data 4.3,-2.6,-1.2,2
data 4.3,-2.8,-.8,2
data 4.3,-2.95,-.4,2
data 4.3,-3,0,2
data 4.3,-2.95,.4,2
data 4.3,-2.8,.8,2
data 4.3,-2.6,1.2,2
data 4.3,-2.2,1.6,2
data 4.3,-1.8,1.8,2
data 4.3,-1.4,1.95,2
data 4.3,-1,2,2
data 4.3,-.6,1.95,2
data 4.3,-.2,1.8,2
data 4.3,.2,1.6,2
data 4.3,.6,1.2,2
data 4.3,.8,.8,2
data 4.3,.95,.4,2
data 4.3,1,0,2
//*** spinner
data 4,-1.1,0,3
data 4.59,-1,0,2
data 4,-.91,0,2
data 4,-1,.09,3
data 4.59,-1,0,2
data 4,-1,-.11,2
//*** axis lines
data -9,-4,-9,3
data 6,-4,-9,2
data 1,-3,-9,3
data 2,-2,-9,2
data 1,-2,-9,3
data 2,-3,-9,2
data -9,-4,-9,3
data -9,4,-9,2
data -7.5,3,-9,3
data -7.5,3.5,-9,2
data -8,4,-9,2
data -7.5,3.5,-9,3
data -7,4,-9,2
data -9,-4,-9,3
data -9,-4,6,2
data -9,-3,4,3
data -9,-3,5,2
data -9,-2,4,2
data -9,-2,5,2
data -9,-4,-9,3 ■
```

UniComal For IBM PC

by UniComal A/S

[UniComal has provided COMAL Today with this description of their IBM PC COMAL system. All our order processing, inventory control, and accounting programs are written and run under UniComal IBM PC COMAL. It is **the** *implementation to compare others to. We are told that has gained an excellent reputation in Europe, and many companies use it for their in house programming.]*

UniComal IBM PC COMAL (called UniComal for short in this article) is a programming language whose popularity has grown dramatically in recent years. It is no longer reserved for an exclusive inner circle of professional programmers. It is now in use by enthusiastic programmers in many areas of endeavour. The goal of UniComal A/S is to provide the most complete and user-friendly programming development tool available.

Structures

UniComal includes all of the structures and facilities which expert programmers expect of a program development tool. It is easy to create flexible, structured programs with multiline versions of structures such as:

```
IF THEN .. ELSE .. ENDIF
CASE .. ENDCASE
REPEAT .. UNTIL
WHILE .. ENDWHILE
LOOP .. EXIT .. ENDLOOP
PROC .. ENDPROC
FUNC .. ENDFUNC
```

In addition, a TRAP .. HANDLER .. ENDTRAP structure makes it possible to capture and handle errors which can occur during program execution. This means crash-proof programs. It is possible to integrate DOS and UniComal programs, since DOS programs can be called directly from UniComal.

Procedures and Functions

UniComal encourages modular programming with independent building blocks. The user's own procedures and functions can be tested quickly and interactively by execution at the command level, both with and without parameter transfer. This makes the programming and debugging phase easier and is in harmony with modern requirements for a program development tool.

It is also possible to define both local and global variables in procedures and functions. Procedures and functions can be defined with or without parameters, which may be specified by value or by reference. Text variables, matrices, reals and integers can all be transferred as parameters. Finally, procedures and functions can be called recursively.

Protected Input Fields

Full screen input makes it easy to build up screen displays with protected input fields of well-defined length. It is easy to allow the user to jump from field to field using the cursor keys, giving your programs a professional finish.

System Package

The SYSTEM package is a convenient collection of useful procedures and functions. These features make it easy to convert text strings between upper and lower case, implement rapid read or write of screen images, define new sorting orders for the character set and more.

File Handling

UniComal supplies a complete and powerful file handling system which can handle sequential and random files in either ASCII or binary format. Matrices and texts can be read and written directly from and to disk in one simple operation. Individual bytes within random files can be addressed individually.

more»

Long Variable Names

It is easy to use long, descriptive names for variables, labels, procedures and functions. You can even use many of the extended IBM character set (foreign language characters), if you wish. This makes your programs easy to read, understand and maintain, without excessive use of comment statements.

String Handling

Handling of text strings is one of UniComal's special strong points. Using the many string handling features available, it is easy to manipulate strings in every imaginable way. And the length of your text strings is limited only by the size of available memory. These features can be an enormous time-saver and can ease program development significantly.

Structured Error Handling

UniComal makes it possible to produce crash-proof programs quickly and easily, while maintaining clarity. This is due to the unique structured error handling block:

TRAP .. HANDLER .. RETRY .. ENDTRAP

Combined with the REPORT statement, this makes it possible for an error message to be sent to a handler structure at a higher level. With the RETRY statement, the statements in which the error occurred (e.g. due to a disk drive door open) can be tried again after supplying a message for appropriate user action.

Graphics

Graphics is a natural part of the UniComal environment, which includes support of the CGA, EGA, and VGA graphics cards as standard. In addition Hercules and Olivetti graphics packages can be supplied as options. In X-Y graphics, most often used in professional applications, it is possible to work with both physical and logical coordinates using the **viewport** and **window** commands. Commands such as **fill** and **paint** for area fill-in are provided. A **shape** command for animation as well as **plottext** for printing text in a selection of sizes and orientations on the graphics screen is standard in the **GRAPHICS** package. Graphics commands, such as **circle, arc,** and **draw**, are supplied. Complete relative graphics is a standard part of the graphics package. Turtle graphics is included as well.

Packages

To ease the task of the programmer, it is possible to use elegant machine language packages at the same level as other UniComal facilities. It is possible for the user to give machine language procedures and functions descriptive names and to use parameter transfer and error handling on a par with all other UniComal structures, providing a simple and elegant solution. The UniComal system provides a well-defined environment for the integration of machine code programs as packages tailor-made to solve special problems. This makes UniComal programs very dependable, easy to read and easy to extend and to update.

Development and Execution Time

The importance of reducing the development time required for new software has become more apparent in recent years. Furthermore, with UniComal, completed programs run at optimum speed [*8 vs. 38 seconds for Mytech on the same computer - see back cover chart*]. Execution speed is not affected by the size of the program, comments or by the number of variables or the length of variable names. And UniComal programs are compact, making efficient use of available memory.

Another important speed factor is screen access, which has been optimized in UniComal,

more»

so that one can choose direct or BIOS screen access, thereby achieving very rapid screen updating. BIOS access would be selected if one desired that UniComal programs be run in various window environments.

A UniComal programmer will also save time because all program lines are syntax-checked as they are entered. This means that errors are located and pointed out with a complete error message, so corrections can be made at once. Furthermore, the program structure is automatically emphasized by indentation of branch and loop structures, procedures and functions.

UniComal programs are convenient to edit. It is easy to scroll listings up or down on the screen. The function and control keys of your computer come in handy as you work. Just a single keypress is required to make room for a new program line or to remove one. The commands **FIND** and **CHANGE** can be used to search for and to search and replace variable names or other parts of your program. All these features are a big advantage, both for the beginner as well as for the professional programmer.

Runtime Compiler

When a program is completed and tested in the interactive UniComal working environment, it can be advantageous to compile the complete program so it can be run directly from DOS on other computers. The UniComal compiler will translate all statements. Each compiled program can be supplied with a serial number and a copyright text when compiled. To ease the compilation process an Automatic Response File can be defined for each task. By compiling programs one can protect the source text of the program and provide effective customer registration. Everyone who is seriously engaged in program development and sale will find the UniComal compiler to be a valuable tool.

Communication Package

The SCOM (Serial COMmunication support, RS-232C) package is an advanced programming tool which makes it possible to handle practically any imaginable RS-232C communications job with up to three RS-232C ports at once. Character conversion tables and hardware/ software handshake (XON/XOFF) can be defined for each port. This package can be used with UniComal programs to control modems, transmission over modem lines to central databases or for computer-to-computer transmission with error checking protocols.

Technical Information

UniComal can be used on a standard IBM PC or compatible under PC-DOS or MS-DOS 2.1 and later versions. But even greater utility will be achieved from UniComal's many features using a fully expanded computer with maximum memory, hard disk, mathematics coprocessor, extended graphics adapter (EGA or VGA card) and other accessories (ports, etc.).

The accuracy of calculations exceeds that which will normally ever be required. Integer variables can be used in the range from -2147483648 to 2147483647 (32 bit), and real variables can be used in the range +/-1E+/-308.

Real numbers are represented in "double IEEE floating point format". In addition, mathematics coprocessor 8087/80287 support is provided as standard. This chip permits mathematical computations to be carried out at high speed.

Program size can exceed 64K, for in UniComal the entire memory space of the computer can be utilized for program packages, external procedures and functions, machine code packages and data. The UniComal compiler permits the inclusion of external procedures, functions and package within the same .EXE file as the main program. [*Editor's note: UniComal*

more»

UniComal IBM PC COMAL - continued

uses 64K sections of memory: 64K max for programs, 64K max for variable/data storage, rest of memory in 64K blocks for packages, and external procedures. Mytech COMAL has only about 30K user memory, and all external procedures and packages must fit into this space as well as the program and variables.]

Current Uses of UniComal

UniComal products are widely used in industry, research, administration and education. Dansk System Elektronik A/S uses UniComal with their PC System Controller for data collection, process control and other tasks. Billund International Airport (approved for category II and III landing approaches) uses UniComal programs for administration and PC System Controllers to collect and treat meteorological data (vital for flight crews and passengers in commercial aircraft landing in bad weather). Prodex ApS produces advanced hi-fi speaker systems for worldwide export using UniComal programming tools, the only system which this user has found adequate to meet the high standards required.

The RAUFOSS-corporation in Norway has developed an advanced press at the Hydroform machining factory for use in laboratories. The product specification required a freely programmable press with subsequent graphical analysis of the pressure and temperature during the process. Due to the need for great flexibility and ease of use, it was natural to select UniComal as the programming language for this job.

Education

It is natural that UniComal is used widely in high schools, technical schools, colleges and universities. UniComal has gained wide acceptance in Europe, especially because of the highly structured nature of the language and operating environment. Thus it is easy to learn to use UniComal, for UniComal programs can use long, descriptive variable and procedure names, making programs exceptionally easy to read and understand. Since UniComal is also used by professional programmers, many students will come to use it later if they pursue careers in industry or research.

Complete Program Development

UniComal IBM PC COMAL 2.1 (sale price $395) includes three disks (master system disk, tutorial disk, and supplemental programs disk), a detailed reference book, and tutorial book packaged in a special UniComal Doc Box.

The PLUS option ($200 sale price) adds the **compiler** and communications package with manuals, in a second Doc Box. ■

African Stone Game

by L. W. Zabel

The African stone game, sometimes called Awari, is an ancient strategy game originally played with stones arranged in small pits in the ground. The only other computer version of this game, of which I am aware, plays a single player against a not very skillful computer. I have taken a different approach in which two players, A and B, play against each other. The function of the computer is to display the pits and the number of stones in each as well as move the stones according to the game rules.

The rules are displayed on the screen if the players so desire. The more stones in the pits at the start of the game, the more difficult the game becomes. Beginners should use 3 and experts, I mean experts, may use 6. Use your own judgement.

Error trapping has been employed where possible. The stone movements can become quite complex and I have tried to cover every eventuality up to and including as may as 19 stones in any one pit. If any player detects an error in the stone distribution, I will appreciate hearing of the details so that I can make program corrections.

[Editor's note: the <u>instructions</u> procedure has been removed from the listing below since this article provides the instructions.]

Instructions

This game is played using pits in the ground and stones as markers. The computerized game uses boxes on the screen to simulate pits and the number of stones is indicated in the center of the boxes. The counting and moving of the stones is all done by the computer. Two rows of six boxes will appear on the screen. At either end of the rows will be a larger **HOME** box. Each small box has an identifying number in the upper left corner. The number of stones in each box will appear in the center of each box. The object of the game is to collect more stones in your **HOME** box than your opponent. The player to move first is selected by chance and is displayed on the screen.

The first player selects a box by entering its identifying number. Be sure that you use the box number, not the number of stones. When «*return*» is pressed, all of the stones in the selected box are distributed, one to a box in a counter clockwise direction including the player's **HOME** box, but not including his opponent's **HOME** box. If the last stone lands in the player's **HOME** box, he receives a free turn. If the last stone lands in an empty box on the player's side, all of the opponent's stones directly opposite the previously empty box are added to the player's **HOME** box. The game ends when a player's six boxes are all empty. At this time, all of the stones in the opponent's six boxes are added to the opponent's **HOME** box.

The players will be asked for the number of stones to be placed in each box at the start of the game. Beginners should select 3 stones per box, intermediate players should select 4 or 5, but leave 6 for the experts.

[Editor's note: Stones works unchanged in CP/M COMAL and UniComal IBM PC COMAL. It does not work on the C64 COMAL cartridge because it uses 80 column screen output.]

```
DIM a(7), b(7)
RANDOMIZE
LOOP
  begin
  start'game
  IF RND(0,1) THEN player(b(),a(),"Player B")
  WHILE NOT end'game DO
    player(a(),b(),"Player A")
    IF NOT end'game THEN player(b(),a(),
    "Player B") // wrap line
  ENDWHILE
```

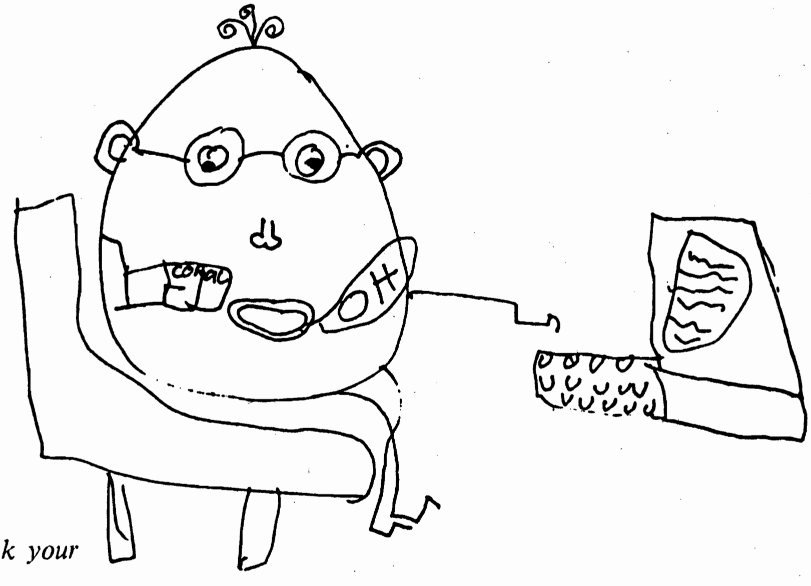
more»

```
    end'routine
  ENDLOOP
  //
  PROC begin
    PAGE
    FOR i:=1 TO 7 DO
      b(i):=0
      a(i):=0
    ENDFOR i
    PRINT AT 10,18: "Welcome to the African
    Stone Game" // wrap line
    INPUT AT 22,23: "Press <RETURN> to
    continue": a1$ // wrap line
    draw'board
  ENDPROC begin
  //
  PROC draw'board
    PAGE
    PRINT AT 7,9:   " ______   ______   ______   ______ "
    PRINT AT 8,9:   "|      | |6     | |5     | |4     |"
    PRINT AT 9,9:   "|HOME  | |      | |      | |      |"
    PRINT AT 10,9:  "|      | |______| |______| |______|"
    PRINT AT 11,9:  "|      |                           "
    PRINT AT 12,9:  "|      |  ______   ______   ______ "
    PRINT AT 13,9:  "|      | |1     | |2     | |3     |"
    PRINT AT 14,9:  "|      | |      | |      | |      |"
    PRINT AT 15,9:  "|______| |______| |______| |______|"
    PRINT AT 7,41:  " ______   ______   ______   ______ "
    PRINT AT 8,41:  "|3     | |2     | |1     | |  A   |"
    PRINT AT 9,41:  "|      | |      | |      | |HOME  |"
    PRINT AT 10,41: "|______| |______| |______| |      |"
    PRINT AT 11,41: "                           |      |"
    PRINT AT 12,41: " ______   ______   ______  |      |"
    PRINT AT 13,41: "|4     | |5     | |6     | |      |"
    PRINT AT 14,41: "|      | |      | |      | |      |"
    PRINT AT 15,41: "|______| |______| |______| |______|"
    PRINT AT 6,37: "B Side"
    PRINT AT 17,37: "A Side"
  ENDPROC draw'board
  //
  PROC print'numbers
    PRINT AT 12,12: USING "###": b(7)
    FOR x:=1 TO 6 DO
      PRINT AT 9,67-8*x: USING "###": b(x)
    ENDFOR x
    FOR x:=1 TO 6 DO
      PRINT AT 14,8*x+11: USING "###": a(x)
    ENDFOR x
    PRINT AT 12,67: USING "###": a(7)
  ENDPROC print'numbers
  //
  PROC blank'out
    PRINT AT 19,6: SPC$(70)
    PRINT AT 20,6: SPC$(70)
  ENDPROC blank'out
  //
  PROC start'game
    LOOP
      TRAP
        INPUT AT 19,6: "How many starting
        stones per pit ": num // wrap line
        EXIT WHEN num>0
      HANDLER
        PRINT AT 20,6: "You must enter a
        number!" // wrap line
      ENDTRAP
    ENDLOOP
    FOR i:=1 TO 6 DO
      a(i):=num
      b(i):=num
    ENDFOR i
    print'numbers
    blank'out
  ENDPROC start'game
  //
  PROC player(REF current'player(),REF
  waiting'player(),name$) // wrap line
    REPEAT
      turn'over:=TRUE
      LOOP
        TRAP
          PRINT AT 19,6: name$+", select a pit
          number? ", // wrap line
          INPUT AT 0,0,1: n
          EXIT WHEN n>0 AND n<7
        HANDLER
          NULL
        ENDTRAP
      ENDLOOP
      last'box:=n+current'player(n)
      IF last'box<8 THEN
        FOR i:=n+1 TO last'box DO current
```

more»

African Stone Game - continued

```
      'player(i):+1 // wrap line
      IF last'box=7 THEN
        current'player(n):=0
        turn'over:=FALSE
      ELIF current'player(last'box)=1 THEN
        current'player(7):+waiting'player(
        7-last'box) // wrap line
        waiting'player(7-last'box):=0
      ENDIF
      current'player(n):=0
    ELIF last'box>7 AND last'box<14 THEN
      FOR i:=n+1 TO 7 DO current'player(i):+1
      FOR i:=1 TO last'box-7 DO
      waiting'player(i):+1 // wrap line
      current'player(n):=0
    ELIF last'box>13 AND last'box<21 THEN
      FOR i:=n+1 TO 7 DO current'player(i):+1
      FOR i:=1 TO 6 DO waiting'player(i):+1
      FOR i:=1 TO last'box-13 DO
      current'player(i):+1 // wrap line
      IF last'box=20 THEN
        current'player(n):=0
        turn'over:=FALSE
      ELIF current'player(last'box-13)=1 THEN
        current'player(7):+waiting'player(
        20-(last'box)) // wrap line
        waiting'player(20-(last'box)):=0
      ENDIF
      current'player(n):=0
    ENDIF
    blank'out
    print'numbers
  UNTIL turn'over OR end'game
ENDPROC player
//
PROC end'routine
  FOR i:=1 TO 6 DO
    a(7):=a(7)+a(i)
    a(i):=0
    b(7):=b(7)+b(i)
    b(i):=0
  ENDFOR i
  print'numbers
  blank'out
  PRINT AT 19,6: "Game end - check your
  scores." // wrap line
  INPUT AT 20,6: "Play another game? (Y/N) ":
  a1$ // wrap line
  IF NOT a1$ IN "Yy" THEN
    END "Hope you enjoyed the game. See you
    next time." // wrap line
  ENDIF
ENDPROC end'routine
//
FUNC end'game
  sum:=0
  FOR i:=1 TO 6 DO sum:+a(i)
  IF sum=0 THEN RETURN TRUE
  sum:=0
  FOR i:=1 TO 6 DO sum:+b(i)
  IF sum=0 THEN RETURN TRUE
  RETURN FALSE
ENDFUNC end'game
//
PROC new'page
  PRINT
  PRINT "Press any key to continue"
  WHILE KEY$="" DO NULL
  PAGE
ENDPROC new'page ■
```



Captain COMAL by Rhianon Lindsay - age 9

Epidemic

by Bill Inhelder

Bob McCauley's *voting'game* program on *Today Disk #17* brought to mind a simulation program which was popular in the early days of Commodore Pets and cassette recorders. Working from memory I have tried to reconstruct the major features of the original program with a few additions of my own.

Epidemic on *Today Disk #19* is a graphical simulation of the spread of an illness in an isolated environment. Several travelers introduce a non-lethal illness to the inhabitants of a small island. All the inhabitants are susceptible to the illness which lasts a specified number of days. Each infected inhabitant is contagious during the period of the illness. The disease is spread by close contact. After recovery from the illness the inhabitant cannot contract the illness again and is no longer contagious.

By varying the number of islanders, the number of contagious persons and the duration of the illness, the extent of the epidemic can be observed. This can range from a full-blown epidemic which affects everyone to a mini epidemic affecting only a few islanders before it ceases. A delay procedure has been included in appropriate places to permit a more detailed observation of the spread of the disease.

Each islander may move in any of eight directions or remain in place. The randomly generated moves obey the following conditions:

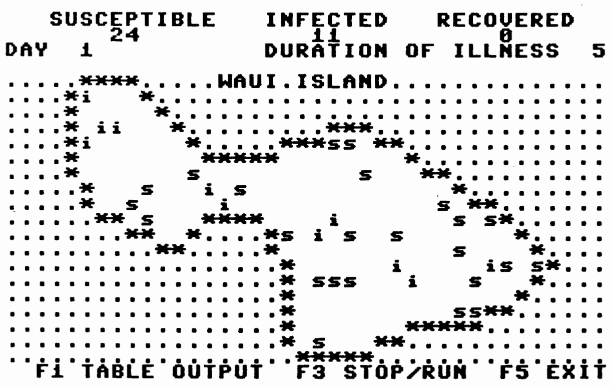
1) if the first randomly generated move is permissible, it is executed; otherwise
2) a second move is randomly generated and is executed if legal; otherwise
3) the inhabitant remains at the same location.

This strategy is designed to give considerable mobility in an uncrowded environment and minimum mobility in a crowded one. The disease spreads more rapidly in crowded conditions.

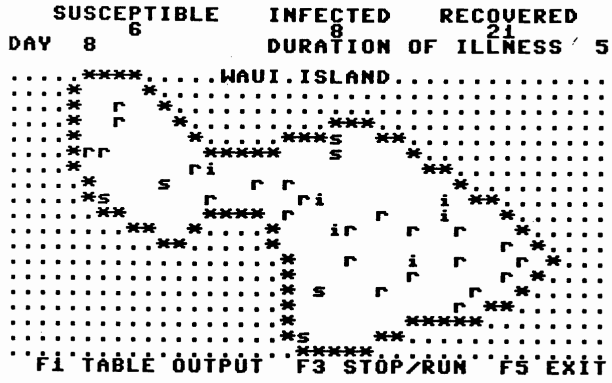
Function keys permit various options while the epidemic is in progress. The «*f3*» key will temporarily halt the progress of the disease. Pressing the key again causes the program to resume. The «*f1*» key will interrupt the program and print out a day-by-day table of the status of the disease.

Once the last infected person has recovered, the program halts and the table may then be printed. Up to 100 days of epidemic history may be simulated.

Experiment with various values to see if you can create different intensities of epidemics. Because the shape of the island tends to isolate inhabitants, it may be somewhat difficult to set up a condition where the entire island succumbs to the illness. ■



```
   SUSCEPTIBLE    INFECTED    RECOVERED
       24            11           0
DAY  1           DURATION OF ILLNESS  5
.....****.....WAUI.ISLAND...............
....*i   *..............................
....*     *.............................
....* ii   *.........***................
....*i      *.....***ss  **.............
....*        *****        *............
....*       s          s    **..........
.....*   s   i s              *.........
.....*  s     i              s **.......
......** s   *****   i        s s*......
........**  *.....*s i s  s       *.....
..........**......*           s    *....
...................*      i     is s*...
...................* sss   i   s   *....
...................*              *.....
...................*          ss**......
...................*       *****........
...................* s   **.............
....................*****...............
  F1 TABLE OUTPUT  F3 STOP/RUN  F5 EXIT
```



```
   SUSCEPTIBLE    INFECTED    RECOVERED
        6             8          21
DAY  8           DURATION OF ILLNESS  5
.....****.....WAUI.ISLAND...............
....*    *..............................
....*  r  *.............................
....*  r   *.........***................
....*       *.....***s   **.............
....*rr      *****   s    *............
....*       ri              **..........
.....*    s     r r           *.........
.....*s      r     ri        i **.......
......**     *****  r    r   i   *......
........**  *.....*   ir   r  r    *....
..........**......*             r *.....
...................*   r   i  r   r *...
...................*       r     r *....
...................* s   r     r  *.....
...................*          r **......
...................*       *****........
...................*s    **.............
....................*****...............
  F1 TABLE OUTPUT  F3 STOP/RUN  F5 EXIT
```

Directory Boxes

by Paul Keck

Although the catalog headers on *Today disks* are very helpful, you don't really need them. I don't like waiting a long time for the directory to scroll by if I want to see something towards the bottom but can't remember the name. So, this program deletes them. It warns you to only use it on a copy of the disk.

[*Editor's note: since Today Disk #9 we have been putting comments in our disk directories. This helps organize the disk, but it makes it difficult to back the disk up by copying all the files (you would get many duplicate entries).*]

```
DIM ke$ OF 1
PAGE
PRINT "This program will remove"
PRINT "those comment boxes from disk"
PRINT "directories. You know the ones-"
PRINT
PRINT "   ┌──────────────┐ "
PRINT "   │ Comment..    │ "
PRINT "   ├──────────────┤ "
PRINT "   │ another one..│ "
PRINT "   │              │ "
PRINT "   └──────────────┘ "
PRINT
PRINT "You should only use this on a copy"
PRINT "of a disk, because you may want to"
PRINT "see those boxes again someday."
PRINT
PRINT "So, put in the disk you want"
PRINT "fixed up and hit RETURN to go"
PRINT "ahead, or any other key to quit."
REPEAT
  ke$:=KEY$
UNTIL ke$<>CHR$(0)
IF ke$=CHR$(13) THEN
  DELETE "0:┌??????????????┐ "
  DELETE "0:│ ??????????????│ "
  DELETE "0:├??????????????┤ "
  DELETE "0:└??????????????┘ "
  DIR "0:*"
ENDIF ■
```

3D Surface Plots

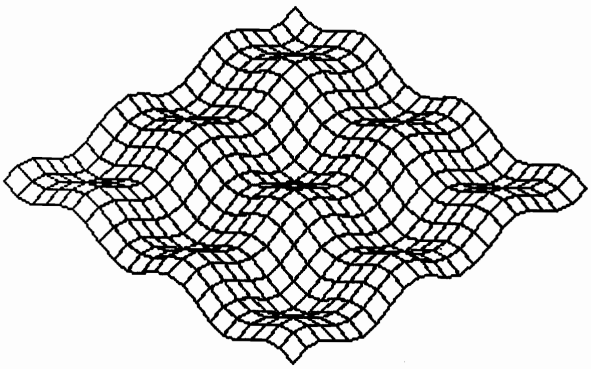
by Sol Katz

3d'surfaces on *Today Disk #19* draws three dimensional surfaces in either a wire frame model or in a solid multi-color filled surface. Several example surfaces are included on the start-up menu. A mesh of 21x21 is the largest reasonable size for the solid surface model, given the resolution of the C-64 in multi-color mode.

The program allows you to choose between 2 perspective view algorithms, called real and estimated. Real gives you more control of the nature of the output and produces a more accurate plot, but takes much longer to run. Estimate is controlled by the variables asp, for aspect, which should range from .25 to 4.0, and h, for viewing height, which should be less than 30. Real accepts values for viewing distance, d (greater than 200), viewing angle above the surface, phi (0 to +/- 89), and rotation angle, theta (0 to +/- 179). These can be thought of as radius, latitude, and longitude, respectively.

Until you become familiar with how the variables affect the plot, I suggest the following starting parameters:

mesh size	- odd values between 5 and 9
PHI	= 30
THETA	= 20
Distance	= 1000
Aspect	= 1
Height	= 10



■

Smarter Reader

by Tim White

Smarter'reader on *Today Disk #19* is an update to *Smart File Reader* from *COMAL Today #16*. The main modification makes the computer and disk drive work in parallel.

When the computer reads text strings from a disk file it enters a brief wait loop between bytes while the drive prepares to send the next byte. This wait is extended when the next byte is the first byte of a sector. As the computer processes the text before writing it to the screen, some time could be saved if this is done while the disk drive is looking for a new sector. We can take advantage of the fact that as soon as the last byte of one sector is read, the disk drive reads the next sector into one of its buffers. The way we use this is information is to get bytes in groups of 254 (the number of data bytes in a sector). When the 254th byte is sent, the disk immediately goes off to fill its buffer with the next sector and the computer goes off to do its text formatting.

The program stores its formatted text strings in its own buffer so it can keep itself and the disk drive working when the screen is being read. It runs through a hierarchy of jobs, first getting and formatting text from the disk, then writing formatted text from the internal buffer to the screen until the first screenful is written. Finally it checks if the user wants to scroll the screen and, if so, starts to scroll at a high priority until it reaches the end of the buffer or there are no more requests to scroll. Scrolling is controlled by the «*crsr*» up and down keys or «*space*». When the whole disk file is read (or the formatted text buffer is full) the program closes the file and devotes all of its time to screen scrolling.

When the user has seen enough of the text, pressing «*stop*» brings up the filename prompt. «Q» causes the program to terminate; entering a new filename causes that file to be displayed. ■

1541 Aligner

by Norman Parron

I was very interested in the 1541 Disk Alignment program by Craig Van Degrift and started using it immediately. But it did lack a few refinements. In my business I wanted some way of making a record of the head response to give to the customer. After seeing the great samples of plots done on the 1520 plotter, I decided to modify the program to incorporate a plotter output routine. I borrowed Kevin Quiggle's plotter routines and added them to the *1541'alignment*. Then I added the **plotter'out** procedure and modified the whole program to suit myself. *1541'alignment* is on *Today Disk #19*.

I've been using this program for two months. I'm very pleased with the results. A few words of caution. I built the signal adaptor according to the instructions given and the response on the screen graph is negative for proper alignment. I fixed this by putting a negative gain in the program. This corrects the graph to a positive plot for correct alignment and it also gives a better response.

This program does not completely replace the original one. I have removed more then half of the original program because once I got the information out of it, I no longer needed it. I threw it out to make my program smaller and faster loading. I align many drives a week and find this program very useful. I've used it while monitoring the response with an oscilloscope and the alignment is made a lot quicker and more accurately with the program then with the oscilloscope.

Further reference:

1541 Disk Alignment Update, COMAL Today #9, page 34
1520 Plotter Driver Procedures, COMAL Today #7, page 62 ■

Value

COMAL Today has released several val or value functions for COMAL 0.14. *COMAL Today #15* has one on page 6 which only handles integers and another on page 49 which uses the disk drive. The topic surfaced again recently while doing the **COMAL 0.14 Power Driver**. To make a disk loaded version of COMAL more compatible with complete COMAL 2.0 systems, it was decided to add a val function (as well as several other missing commands).

In order to make it easier to code the machine language routine, it was decided to first work out the algorithm in COMAL. The function below is named value. This avoids the name conflicts that would occur with COMAL 2.0 or **Power Driver** if val was used. The first thing the function does is copy the string containing the number to a second string variable and appends an exclamation mark to it. (Note, it is not possible to append to the original string parameter **number$** because string parameters are dimensioned to their exact initial length.) The extra character is helpful because it is easier to test for an invalid character than to test for the *end* of a string.

The number is then tested for being positive or negative. Then the integer part of the number is converted from the string to a number. If there is a decimal part, the number is still multiplied by 10 and gets an integer from 1 to 9 added to it for each digit to the right of the decimal point. (Working in integers eliminates round off error). Next the routine checks for an «e» indicating scientific notation. Finally, the number is multiplied or divided by ten enough times to adjust for the decimal and or scientific notation.

Note, the above process is terminated as soon as a character which cannot be part of the number is found. It differs from the VAL function in COMAL 2.0 which reports an error if the string does not contain a valid number. Value returns a zero.

```
FUNC value(number$) CLOSED
  DIM num$ OF LEN(number$)+1
  sign:=1; ptr:=1; v:=0; e:=0
  num$:=number$+"!"
  WHILE num$(ptr:ptr)=" " DO ptr:+1
  IF num$(ptr:ptr)="-" THEN
    sign:=-1; ptr:+1
  ELIF num$(ptr:ptr)="+" THEN
    ptr:+1
  ENDIF
  WHILE digit DO
    v:=v*10+ORD(num$(ptr:ptr))-ORD("0")
    ptr:+1
  ENDWHILE
  IF num$(ptr:ptr)="." THEN
    ptr:+1
    WHILE digit DO
      v:=v*10+ORD(num$(ptr:ptr))-ORD("0")
      ptr:+1; e:-1
    ENDWHILE
  ENDIF
  IF num$(ptr:ptr) IN "eE" THEN
    ptr:+1; e2:=0; e2s:=1
    IF num$(ptr:ptr)="-" THEN
      e2s:=-1; ptr:+1
    ELIF num$(ptr:ptr)="+" THEN
      e2s:=1; ptr:+1
    ENDIF
    WHILE digit DO
      e2:=e2*10+ORD(num$(ptr:ptr))-ORD("0")
      ptr:+1
    ENDWHILE
    e:+e2*e2s
  ENDIF
  IF e>0 THEN
    FOR x:=1 TO e DO v:=v*10
  ELIF e<0 THEN
    FOR x:=-1 TO e STEP -1 DO v:=v/10
  ENDIF
  RETURN sign*v
  //
  FUNC digit
    RETURN num$(ptr:ptr) IN "0123456789"
  ENDFUNC digit
ENDFUNC value ■
```

Extra Programs

1520 Stereo Plane

Herbert G. Denaci showed how to make orthogonal and perspective projections of a 3-Dimensional object in *COMAL Today #13*. He now adds that a stereograph can be made by two successive perspective projections of the object at small increments of angle and displacement. This is what your eyes do when viewing an object at reasonably close distances. A stereoscope can then be used to view the stereograph to obtain the enhanced depth perception or "stereo" effect. Stereo movies produce the effect by photographing in two colors and then having the viewer use blue and red glasses to view the stereographic pictures.

Using the Commodore 1520 printer/plotter allows red and green views for the stereo effect. Green and red glasses can then be used to view the stereograph. Both COMAL 2.0 and 0.14 versions of the program *1520stereo'plane* on *Today Disk #19* produce examples of the stereo illusion of depth at the suggested stereo angles and displacements. Of course, you should try other values of the various parameters for suitable stereo effects. Note that orthogonal projections do not produce suitable results.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Lite Byte

Richard and Todd Shagott wrote a computerized version of *Lite Bright* for COMAL 2.0. Its grid has a resolution of 54 *pegs* wide by 50 *pegs* long, and there are no plastic pegs to loose, misplace, or swallow (a big plus).

Lite'byte on *Today Disk #19* requires a joystick to move the cursor and place colored pegs on the grid. The «a» key can be used to access the menu to load, save, or erase the picture, or display the disk directory. *Lite'byte* comes with two additional files: *hrg.grid* and *hrg.hello*. These pictures should help you get started.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Cave Warrior

Cave'warrior is an action game by N. Bakker on *Today Disk #19*. It requires a joystick in port 2 to control your space ship. Try to shoot your way to the alien city and destroy it.

The program uses the **Irq** package from *COMAL Today #15* to control the sprites. A new package, **Splitscleft**, is used to scroll the screen to the left. The game features sound affects.

Note: this program uses the cassette buffer. To use it on a C128 with Super Chip, you must discard the **C128** package:

```
USE c128
discard'c128
RUN "cave'warrior"
```

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Ghosts

In this game from the Dutch COMAL User's Group, it is your duty to bury your friend. This is not easy because you have to fetch the shovel, dig the grave, and get the coffin while witches try to drop pitch forks on you. Don't despair, after your task is done, your friend *may* go to heaven. *Ghosts* is on *Today Disk #19*.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Mandelbrots

Jim Frogge sent us a Mandelbrot program which uses Super Chip on the C128 to speed things up a bit. Ray Carter sent us a faster version of Ted Groszkiewicz's Mandelbrot program from *COMAL Today #18*. Both programs are on *Today Disk #19*. ■

Power Driver Keywords

++ after keyword means added by **Power Driver**
-- means both COMAL 0.14 and Power Driver

// -- allows comments in a program
// anything typed here

ABS -- gives the absolute value
ABS(«numeric expression»)
PRINT ABS(standard'number)

AND -- logical AND
«expression» AND «expression»
IF number>0 AND number<100 THEN

APPEND -- start at end of file for writing
OPEN [FILE] «file#»,«filename»,APPEND
OPEN FILE 2,"test",APPEND

AT ++ see INPUT AT, PRINT AT

ATN -- arc tangent
ATN(«numeric expression»)
PRINT ATN(num1+num2)

AUTO -- automatic line numbering
AUTO [«start line»][,«increment»]
AUTO 9000

BACK -- move turtle backwards
BACK «length»
BACK 50

BACKGROUND -- set background color
BACKGROUND «color number»
BACKGROUND 2 // red

BASIC -- exit COMAL to BASIC, see also BYE

BITAND ++ bitwise AND
«argument» BITAND «argument»
show(bnum BITAND %00001000)

BITOR ++ bitwise OR
«argument» BITOR «argument»
PRINT (bnum BITOR flag)

BITXOR ++ bitwise XOR
«argument» BITXOR «argument»
bnum=(num1+num2) BITXOR %10000000

BORDER -- set the screen border color
border «color number»
border 0 // black

BYE ++ exit COMAL to BASIC
BYE

CASE -- multiple choice decisions
CASE «control expression» [OF]
CASE reply$ OF

CAT -- gives disk directory, see also DIR
CAT [«drive num»]
CAT 0

CHAIN -- load & run program on disk
CHAIN «filename»
CHAIN "menu"

CHR$ -- gives the character specified
CHR$(«numeric expression»)
PRINT CHR$(num)

CLEAR -- clear the graphics screen
CLEAR

CLOSE -- closes files
CLOSE [[FILE] «filenum»]
CLOSE FILE 2

CLOSED -- all proc/func variables local
PROC «procname»[(params)] [CLOSED]
FUNC «funcname»[(params)] [CLOSED]
PROC newpage(header$) CLOSED

CON -- continue program execution after STOP
CON

COS -- cosine
COS(«numeric expression»)
PRINT COS(number)

more»

CURCOL ++ returns the cursor column position
CURCOL
column:=CURCOL

CURROW ++ returns the cursor row position
CURROW
row:=CURROW

CURSOR ++ positions the cursor
CURSOR «row»,«col»
CURSOR 5,1

DATA -- provides data for a READ
DATA «value»{,«value»}
DATA "Sam",34,"Fred",22,"Gloria",46

DATACOLLISION -- sprite/data collision
DATACOLLISION «sprite#»,«reset colisn flag?»
DATACOLLISION 3,true

DEFINE -- set up a sprite image for later use
DEFINE «shape#»,«64 byte string def»
DEFINE 4,shape$

DEL -- deletes lines
DEL «range»
DEL 460 - 580

DELETE -- deletes a file from disk
DELETE «filename»
DELETE "test"

DIM -- reserve string/numeric array space
DIM «string var» OF «max char»
DIM «str array»(«index») OF «max char»
DIM «array name»(«index»)
DIM players$(1:4) OF 10
DIM expenses(months,categories)

DIR ++ display directory of disk, see also CAT
DIR [«filename»]
DIR "database."*

DIV -- division with integer answer
«dividend» DIV «divisor»
result=guess DIV count

DO -- see FOR and WHILE

DRAWTO -- draw a line from current point
DRAWTO «x coord»,«y coord»
DRAWTO 50,80

EDIT -- list and edit lines without indentations
EDIT [«range»]
EDIT 550-
EDIT

ELIF -- shortened ELSE IF condition
ELIF «expression» [THEN]
ELIF reply$ IN "YyNn" THEN

ELSE -- alternative in IF structure
ELSE

END -- halt program
END

ENDCASE -- end of CASE structure
ENDCASE

ENDFOR -- end of FOR structure
ENDFOR [«control variable»]
ENDFOR sides
ENDFOR count#

ENDFUNC -- end of function
ENDFUNC [«function name»]
ENDFUNC even

ENDIF -- end of IF structure
ENDIF

ENDPROC -- end of procedure
ENDPROC [«procedure name»]
ENDPROC show'item

ENDWHILE -- end of WHILE structure
ENDWHILE

ENTER -- retrieve ASCII program lines
ENTER «filename»
ENTER "testing"

more»

EOD -- End Of Data flag
EOD
WHILE NOT EOD DO

EOF -- End Of File flag
EOF(«filenum»)
WHILE NOT EOF(infile) DO

ESC -- STOP key pressed flag
ESC
IF ESC THEN

EXEC -- execute a procedure
[EXEC] «procname»[(«parameter list»)]
show'item(number)

EXP -- natural log e to n
EXP(«numeric expression»)
PRINT EXP(number)

FALSE -- predefined value equal to 0
FALSE
ok:=FALSE

FILE -- see INPUT, PRINT, READ, WRITE

FILL -- fills in area with current color
FILL «x coord»,«y coord»
FILL 50,80

FOR -- start of FOR loop structure
FOR«var»:=«#»TO«#»[STEP«#»]DO[«statmnt»]
FOR x:=10 TO 1 STEP -1 DO PRINT x
FOR player#:=1 TO max# DO

FORWARD -- move turtle forward
FORWARD «length»
FORWARD 100

FREE ++ returns available program memory
FREE
PRINT FREE
IF FREE>2000 THEN max:+300

FULLSCREEN -- fullscreen graphics (f5)
FULLSCREEN

FUNC -- start of a multi-line function
FUNC «name»[(«parm»)] [CLOSED]
FUNC call'answered

GET$ ++ returns # of characters from open file
GET$(«filenum»,«# of characters»)
text$=GET$(2,16)

GETCOLOR -- returns color of specified pixel
GETCOLOR(«x coord»,«y coord»)
print GETCOLOR(50,80)

GOTO -- go to line after specified label
GOTO «label name»
GOTO jail

HIDESPRITE -- turn off specified sprite
HIDESPRITE «sprite#»
HIDESPRITE 2

HIDETURTLE -- make turtle invisible
HIDETURTLE

HOME -- put the turtle in its home position
HOME // *x=160 & y=99 is home*

IDENTIFY -- assign a shape to a sprite
IDENTIFY «sprite#»,«shape#»
IDENTIFY 2,14 **(sprite 7 is the turtle)**

IF -- start of conditional IF structure
IF «condition» THEN [«statement»]
IF reply$ IN "yYnN" THEN

IN -- locate string1 within string2
«string1» IN «string2»
IF guess$ IN word$ THEN winner

INKEY$ ++ return one character from keyboard
INKEY$
CASE INKEY$ OF

INPUT -- input from keyboard or file
INPUT FILE «file#»[,«rec#»]: «var list»
INPUT [«prompt»:] «vars»
INPUT "ZIP CODE: ": zip'code,

more»

INPUT AT ++ input from keyboard at location
INPUT AT «row»,«col»: [«prompt»:] «vars»
INPUT AT 0,10: "Last name: ":last'name$
INPUT AT 5,15: age

INT -- nearest integer (less than or equal)
INT(«numeric expression»)
tally:+INT(number)

KEY$ -- scans keyboard & returns key typed
KEY$
WHILE KEY$<=CHR$(0) DO NULL

LABEL -- assign label name to the line
[LABEL] «label name»:
quick'quit:

LEFT -- turn turtle left
LEFT «degrees»
LEFT 90 // a right angle

LEN -- gives the length of a string
LEN(«string expression»)
length=LEN(text$)

LINEFEED -- set linefeed to printer
LINEFEED «+/-»
LINEFEED + //linefeeds on
LINEFEED - //linefeeds off

LIST -- list program lines
LIST [«range»] [«filename»]
LIST "myprog.lst"
LIST 380-450 "readrec.proc"

LOAD -- load a program from disk
LOAD «filename»
LOAD "menu"

LOG -- natural logarithm of n
LOG(«numeric expression»)
PRINT LOG(number);

MOD -- remainder of division (modula)
«dividend» MOD «divisor»
color=number MOD 16

MOVETO -- change graphics location
MOVETO «x coord»,«y coord»
MOVETO 50,80

NEW -- clears program from memory
NEW

NEXT -- converted to ENDFOR, see ENDFOR

NOT -- logical NOT
NOT «condition»
IF NOT ok THEN

NULL -- does nothing
NULL
WHILE KEY$<>"c" DO NULL

OF -- see DIM and CASE

OPEN -- open a file
OPEN [FILE] «file#»,«filename»,«type»
OPEN FILE 2,"scores",READ
OPEN FILE 4,"",UNIT 4,7,WRITE
OPEN FILE 3,"subs.ran",RANDOM 50

OR -- logical OR
«condition» OR «condition»
IF reply$<"a" OR reply$>"z" THEN

ORD -- ASCII (ordinal) value of char
ORD(«string expression»)
a:=ORD("a")

OTHERWISE -- default for CASE
OTHERWISE

PAGE ++ clearscreen / formfeed
PAGE

PASS -- send command to disk drive
PASS «command$»
PASS "i0"

PEEK -- look at memory location
PEEK(«memory address»)
device=PEEK(4839)

more»

PENCOLOR -- set turtle drawing color
PENCOLOR «color number»
PENCOLOR 2 // red

PENDOWN -- put pen down, turtle draws
PENDOWN

PENUP -- pick pen up, turtle does not draw
PENUP

PI ++ value of pi
PI
PRINT "Value of PI is";PI

PLOT -- plot a point in current color
PLOT «x coord»,«y coord»
PLOT 50,80

PLOTTEXT -- put text on graphics screen
PLOTTEXT «x coord»,«y coord»,«text$»
PLOTTEXT 0,24,"press space to continue"

POKE -- change contents of memory location
POKE «memory address»,«contents»
POKE 4839,13

PRINT -- print items to screen/printer/file
PRINT [USING «format»:] «list»
PRINT [FILE «#»[,«rec»]:][USING «frm»:]«list»
PRINT FILE 2: text$

PRINT AT ++ print to screen location
PRINT AT «row»,«col»: [USING «frm»:]«list»
PRINT AT 24,1: "Press any key to continue"

PRIORITY -- data priority over sprite?
PRIORITY «sprite#»,«data priority?»
PRIORITY 2,false

PROC -- start of multi-line procedure
PROC «name»[(«parm»)] [CLOSED]
PROC readrec(number)

RANDOM -- random access disk file
OPEN FILE «file#»,«filename»,RANDOM «len»
OPEN FILE 2,"subs",RANDOM 88

RANDOMIZE ++ seeds rnd number generator
RANDOMIZE [«seed»]
RANDOMIZE
RANDOMIZE 9

READ -- read data from DATA line or file
READ [FILE «file#»[,«rec#»]:] «var list»
OPEN [FILE] «filenum»,«filename»,READ
READ name$,age
READ FILE 2,record: name$,adr$,city$,st$
OPEN FILE 2,"scores.dat",READ

REF -- parm var used in reference (alias)
REF «var»
PROC alter(REF text$) CLOSED

REM -- remarks, REM is converted to //
// [«text»]
// sprite data

RENUM -- renumber program
RENUM [«target start»][,«increment»]
RENUM 9000,1
RENUM 100

REPEAT -- start of REPEAT structure
REPEAT

RESTORE -- reuse DATA with READ
RESTORE

RETURN -- returns value of a function
RETURN [«value»]
RETURN TRUE

RIGHT -- turn turtle right
RIGHT «degrees»
RIGHT 180 // reverse direction

RND -- random number
RND («start num»[,«end num»])
dice=RND(1,6)+RND(1,6)
probability=RND(0)

RUN -- run program in memory or on disk
RUN

more»

SAVE -- store program to disk
SAVE «filename»
SAVE "zombies"

SCAN ++ scan for correct structures
SCAN

SELECT -- choose output location
SELECT [OUTPUT] «type»
SELECT "lp:" //printer

SETEXEC -- tells system to list EXEC
SETEXEC «+/-»
SETEXEC + //show EXEC keyword
SETEXEC - //omit EXEC keyword (default)

SETGRAPHIC -- turn on graphics screen
SETGRAPHIC [«type»]
SETGRAPHIC 0 // hi-res screen
SETGRAPHIC 1 // multi-color
SETGRAPHIC // use previous

SETHEADING -- set turtle heading
SETHEADING «degrees»
SETHEADING 180

SETTEXT -- turn on text screen (**f1**)
SETTEXT

SETXY -- set turtle x, y coordinates
SETXY «x coord»,«y coord»
SETXY 50,80

SGN -- -1 if neg, 0 if 0, 1 if pos
SGN(«numeric expression»)
flag=SGN(number)

SHOWTURTLE -- make turtle visible
SHOWTURTLE

SIN -- gives sine
SIN(«numeric expression»)
PLOT(SIN(num),y)

SIZE -- report on free memory
SIZE

SPC$ ++ returns # of spaces specified
SPC$(«number of spaces»)
PRINT SPC$(39)

SPLITSCREEN -- 2 text lines above graphics
SPLITSCREEN // or use (**f3**)

SPRITEBACK -- set 2 multicolor sprite colors
SPRITEBACK «color1»,«color2»
SPRITEBACK 2,6 //red & blue

SPRITECOLLISION -- sprite/sprite collsn
SPRITECOLLISION «sprite#»,«reset colisn flg»
SPRITECOLLISION 2,false

SPRITECOLOR -- set color of sprite
SPRITECOLOR «sprite#»,«color number»
SPRITECOLOR 2,6 //sprite 2 red

SPRITEPOS -- position sprite at x,y location
SPRITEPOS «sprite#»,«x coord»,«y coord»
SPRITEPOS 2,160,99// x=160 & y=99 position

SPRITESIZE -- set sprite size (expand or not)
SPRITESIZE «sprite#»,«x expand»,«y expand»
SPRITESIZE 2,true,true //double size

SQR -- gives square root
SQR(«numeric expression»)
root=SQR(number)

STATUS -- returns disk drive status message
STATUS
disk'err$:=STATUS$

STEP -- increment FOR loop by this amount
STEP «numeric expression»
FOR x=1 TO max STEP 2 DO

STOP -- halt program execution
STOP

STR$ ++ converts number into string
STR$(«number»)
zip$=STR$(number)

more»

SYS -- call machine language subroutine
SYS «address»
SYS 838

TAB -- move cursor to specified column
TAB(«column number»)
PRINT TAB(col), name$

TAN -- gives tangent
TAN(«numeric expression»)
PRINT TAN(number)

THEN -- part of IF structure
THEN
IF ok THEN

TIME ++ set or return time (in 1/60 sec)
TIME [«jiffies»]
PRINT "Seconds ="; TIME/60
TIME 0

TO -- part of FOR structure
«start num» TO «end num»
FOR x:=1 TO 4 DO

TRAP -- enable/disable the «stop» key
TRAP ESC «+/-»
TRAP ESC - //disable «stop» key
TRAP ESC + //enable «stop» key

TRUE -- predefined value of 1
TRUE
RETURN TRUE

TURTLESIZE -- set turtle size (0 to 10)
TURTLESIZE «size»
TURTLESIZE 6

UNIT -- specifies unit in OPEN statement
OPEN FILE «filenum»,«name»[,UNIT «unit#»
[,«secondary address»]][,«type»] // wrap
OPEN FILE 2,"database",UNIT 9,READ

UNTIL -- end of REPEAT loop
UNTIL «condition»
UNTIL reply$="q"

USING -- formatted output
PRINT USING «format»: «var list»
PRINT USING "##> $###.##": x, cash(x)

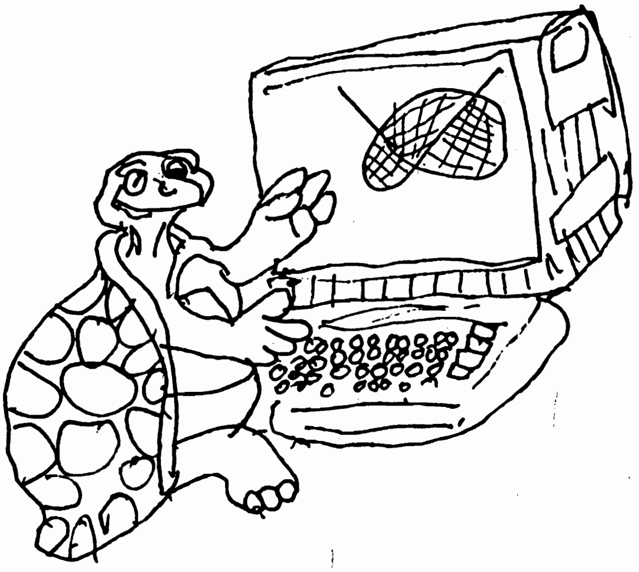
VAL ++ returns numeric value of string
VAL(«numeric string»)
age=VAL(reply$)

WHEN -- choice in CASE structure
WHEN «list of values»
WHEN "Jan","jan"

WHILE -- start of WHILE structure
WHILE «expression» [DO] [«statement»]
WHILE NOT EOF(infile) DO process

WRITE -- write to a file
WRITE FILE «file#»[,«rec#»]:«var»
OPEN [FILE] «filenum»,«filename»,WRITE
WRITE FILE 2: name$
OPEN FILE 3,"scores",WRITE

ZONE -- tab interval
ZONE [«tab interval»]
ZONE 5 ■



Calvin COMAL by Rhianon Lindsay - age 9

Order Form

ORDER FORM

Subscriber #______

(more on other side)

Oct 1987 - Prices subject to change

(only subscribers get lower prices)

Prepaid orders only, US Dollars, or charge:

Name:______________________________

Street:______________________________

City/St/Zip:______________________________

VISA/MC #:______________________________exp date:_______Signature:________________

Circle price of items wanted: (first is DocBox pages price, second-subscriber price, third-list price - DocBox is pages only)

BOOKS - add $3 shipping per book (overseas extra) (Canada add $1 extra shipping per book)

DocBox	Sub price	List price	Item description - disks are Commodore format - DocBox is pages only
	$9.95	$14.95	Doc Box, mini 3 ring binder & slip case, cloth bound, quality D-ring style **One Doc Box is required to hold books you buy as Doc Box pages!!!** (see special below)
$18.95	-.-	$20.95	New-Common COMAL Reference, Len Lindsay (formerly COMAL Cross Reference) *General* || Detailed reference book for COMAL 2.0 implementations in USA (CP/M, Mytech, UniComal)
$17.95	-.-	$19.95	New-CP/M COMAL Package Guide, Richard Bain *CP/M COMAL* || Guide to making your own packages for CP/M COMAL
$17.95	-.-	$19.95	COMAL 2.0 Packages, Jesse Knight, *108 pages with disk* *C64 2.0 Advanced* || How to write a package in Machine Code; includes C64 comsymb & supermon
$17.95	$17.95	$19.95	Packages Library Volume 1, David Stidolph, *76 pages with disk* *C64 2.0* || 17 example packages ready to use, many with source code, plus the Smooth Scroll Editor
$17.95	-.-	$19.95	Packages Library Volume 2, *68 pages with disk* (in stock) *C64 2.0* || 24 example packages ready to use, most with source code
$13.95	$14.95	$15.95	Graph Paper, Garrett Hughes, *52 pages with disk* (now in stock) *C64 2.0* || Function graphing system for C64 COMAL 2.0. The program can't be LISTed.
$18.95	-.-	$20.95	COMAL Collage, Frank & Melody Tymon, *168 pages with disk* (now in stock) *C64 2.0* || Graphics and Sprites tutorial with many full sized example programs
$16.95	-.-	$18.95	3 Programs in Detail, Doug Bittinger, *book and disk* (now in stock) *C64 2.0* || A step by step guide to creating: Blackbook, Home Accountant, BBS
$9.95	-.-	$11.95	New-Today Tutorials, From COMAL Today #1-17, updated as needed (due Dec 1987) *General* || Includes explanations of structures, procedures, functions, and string handling
$9.95	-.-	$11.95	New-Today Tips & Notes, Collection of programming tips and notes (due Dec 1987) *C64 0.14 & 2.0* || Taken from COMAL Today issues #1-17, updated as needed
$4.95	$4.95	$6.95	Cartridge Graphics & Sound, Captain COMALs Friends, *64 pages* *C64 2.0* || A reference book to the 11 packages built into the C64 COMAL 2.0 cartridge
$-.-	-.-	$-.-	Cartridge Tutorial Binder, Frank Bason & Leo Hojsholt, *320 pages with disk* *C64 2.0* || Tutorial from Commodore Denmark - Discontinued due to import problems.
-.-	$4.95	$6.95	COMAL Today - The Index, Kevin Quiggle, *52 pages with disk.* *General* || 4,848 entry index to COMAL Today issues 1-12, full data disk files & reader program
$12.95	-.-	$14.95	3rd printing-Library of Functions & Procedures, Kevin Quiggle, *80 pages with disk* *C64 0.14* || Reprinted by popular demand. Over 140 procedures & functions ready to use
-.-	$17.95	$19.95	Introduction to Computer Programming with COMAL, J William Leary, *272 pages* *C64 2.0* || Beginners text book, spiral bound, now includes separate answer book
-.-	$16.95	$18.95	COMAL Handbook, Len Lindsay, *479 pages* *C64 0.14 & C64 2.0* || Detailed reference book for C64 COMAL 0.14 and 2.0
-.-	$17.95	$19.95	Foundations in Computer Studies With COMAL, John Kelly, *363 pages* *General* || Beginners text book, Jr/Sr High School level
-.-	$18.95	$20.95	Beginning COMAL, Borge Christensen, *333 pages* (imported from England) *General* || Beginners text book, Elementary School level, written by the founder of COMAL
-.-	$4.95	$6.95	COMAL From A to Z, Borge Christensen, *64 pages* *C64 0.14* || Mini reference guide to C64 COMAL 0.14 by the founder of COMAL
-.-	$12.95	$14.95	Captain COMAL Gets Organized, Len Lindsay, *102 pages with disk* *C64 0.14* || Application tutorial to illustrate benefits of modular programming
-.-	$4.95	$6.95	COMAL Workbook, Gordon Shigley, *69 pages* *C64 0.14* || Companion to the Tutorial Disk, great for beginners, full sized fill in the blank style

---» Add the total for items on this side to the bottom of the other side. Shipping add $3 per book.

Then mail to: COMAL Users Group, USA, Limited, 6041 Monona Drive, Madison, WI 53716

Order Form

ORDER FORM Subscriber #______ Name:______________________________
(more on other side)
Oct 1987 - Prices subject to change Street:______________________________
(only subscribers get lower prices)
Prepaid orders only, US Dollars, or charge: City/St/Zip:______________________________

VISA/MC #:______________________________exp date:________Signature:______________________

To order, circle price or check box as indicated:

NEWSLETTER & DISK SUBSCRIPTIONS (and single issues)

$18.95	(6 issues)	COMAL Today newsletter subscription--> [___]renew [___]start at current issue
$26.95	(10 issues)	(Canada add $1 per issue; Overseas add $5 per issue)
$3.95	each	Backissues COMAL Today->circle issues wanted: 5 6 7 8 9 10 11 12 13 14 15 16 17
$15.95	(subscribers only)	COMAL Yesterday, first 4 issues of COMAL Today, spiral bound
$35.95	(6 disks)	Today Disk subscription--> [___]renew [___]start at current [___]start at #______
$55.95	(10 disks)	
$9.75	each	Today Disk-> circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

SYSTEMS:

Sub price	List price	Item Description - disks are Commodore format unless otherwise stated
$4.00	$5.00	New-CP/M COMAL 2.10 Final Demo disk (the SAVE and ENTER commands are disabled)
$49.95	$69.95	New-CP/M COMAL 2.10 Final Full System, manual & disk in Doc Box (ship $5)
$19.95	$39.95	New-CP/M COMAL RUNTIME System (creates stand alone programs like a compiler)
		(CP/M COMAL & RUNTIME run on the C128 in CP/M mode ... disk is in CP/M format)
$27.95	$29.95	C64 COMAL 0.14 Starters Kit (*4 disks, 2 books, 6 newsletters, more*)-(ship $4)
$128.90	$138.95	C64 COMAL 2.0 Cartridge Deluxe Pak (*cart, superchip, 2 books, 4 disks*) (ship $5)
$89.95	$99.95	C64 COMAL 2.0 Cartridge plain (*no manual, no disks*) (ship $2)
$24.95	$29.95	Super Chip for black C64 COMAL 2.0 Cartridge (*add $5 for installation/testing fee*) (ship $1)
$185.00	$199.95	New-C128 COMAL 2.0 Cartridge (*cartridge, manual, demo disk*) (special order only) (ship $5)
$9.75	$14.95	Commodore PET 32K computer COMAL 0.14 (*disk only*-4040 format)
$595.00	$595.00	UniComal IBM PC COMAL 2.1 (*English manual*) (special order only) (ship $5)
-.-	-.-	*Coming soon ... Apple COMAL! Final Mytech IBM PC COMAL* indefinately delayed

DISK SETS:

$25.95	$29.95	New-Full set of 12 European 2.0 Program disks (3 from England & 9 from Holland) (ship $2)
$14.95	$24.95	All four C64 2.0 Cart Demo Disks (1,2,3,4) (ship $2)
$25.95	$29.95	Full set of eleven 0.14 User Group disks (1,2,3,4,5,6,7,8,9,10,12) (ship $2)
$15.95	$19.95	Full set of four 2.0 User Group disks (11,13,14,15) (ship $2)
$9.75	$10.00	Single User Group Disk; circle which one--> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
$19.95	$24.95	Super Chip On Disk plus Super Chip Programs Disk
$49.95	$99.95	New-Source Code to Super Chip (requires Commodore's Assembler & Packages Library Fixes)

SPECIALTY DISKS ($9.75 each) (check box to order)

[___]DataBases(0.14/2.0)
[___]Font(0.14/2.0)
[___]Games(0.14/2.0)
[___]Modem(0.14/2.0)
[___]New-Full 0.14 Sampler from San Francisco show

[___]TutorialDisk(0.14)
[___]AutoRunDemo(0.14)
[___]ParadiseDisk(0.14)
[___]BricksTutorials(0.14)

[___]Utility #1(0.14)
[___]Utility #2(0.14)
[___]SlideShow#1&2(0.14)
[___]SpanishCOMAL(0.14)
[___]Articles#1(textfiles)

[___]SuperChip Programs
[___]Shareware2.0(3 sides)
[___]Read & Run(2.0)
[___]Math & Science(2.0)
[___]Typing(2.0)

Total $_______ + $______ shipping (this side) -- Shipping is $3 per book
Total $_______ + $______ shipping (other side) -- <u>Minimum shipping charge of $2 per order</u>

Total $_______ + $______ = US$_______ Total Paid (WI add 5% sales tax) (overseas is extra)

Mail to: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call (608) 222-4432
(allow two weeks for checks to clear ... $10 charge for checks not honored by the bank)

COMMON COMAL - LOOPS

```
REPEAT
  «block»
UNTIL «condition»

WHILE «condition» DO
  «block»
ENDWHILE

LOOP
  «block»
  EXIT WHEN «condition»
  «block»
ENDLOOP

FOR «v»:=«start» TO «end» DO
  «block»
ENDFOR «v»
(STEP «amount» is an option)
```

```
REPEAT
  INPUT "Are you done?":reply$
UNTIL reply$ IN "YyNn"

WHILE NOT EOF(2) DO
  READ FILE 2: text$
  PRINT text$
ENDWHILE

LOOP
  INPUT "Score (0=done)?":score
  EXIT WHEN score=0
  WRITE FILE 2: score
ENDLOOP

FOR month:=1 TO 12 DO
  PRINT month'name$(month);
ENDFOR month
```

```
REPEAT
  solve'problem
UNTIL errors>3

WHILE KEY$="" DO
  flash(prompt$)
ENDWHILE

LOOP
  READ FILE 2: name$
  EXIT WHEN name$="*end*"
  PRINT name$
ENDLOOP

FOR x:=-1 TO 1 DO
  READ sign$(x)
ENDFOR x
DATA "neg","zero","pos"
```

COMMON COMAL - DECISIONS

```
CASE «selector» OF
WHEN «choice list»
  «block»
WHEN «choice list»
  «block»
...
OTHERWISE
  «block»
ENDCASE

IF «condition» THEN
  «block»
ELIF «condition» THEN
  «block»
...
ELSE
  «block»
ENDIF
```

```
CASE reply$ OF
WHEN "a","A" // add
  add'member
WHEN "d","D" // delete
  delete'member
WHEN "l","L" // list
  list'member
OTHERWISE // invalid choice
  PRINT "I can't do that."
ENDCASE

IF letter$ IN vowel$
  PRINT "It is a vowel"
ELIF letter$ IN consonant$
  PRINT "It is a consonant"
ELSE
  PRINT "It is not a letter"
ENDIF
```

```
CASE eaten OF
WHEN 0
  PRINT "You might starve"
WHEN 1,2
  PRINT "Not bad"
WHEN 3
  PRINT "Great, I ate 3 too"
OTHERWISE
  PRINT "I won't pay the bill"
ENDCASE

IF subscriber THEN
  PRINT "Subscriber discount";
  price:-2 // subtract 2 dollars
ELSE
  PRINT "Normal order";
ENDIF
PRINT USING "$##.##": price
```

COMMON COMAL - ERROR TRAPPING

```
TRAP
  «block»
HANDLER
  «block»
  (REPORT, ERR, ERRTEXT$)
ENDTRAP
```

```
TRAP
  average:=score DIV number
HANDLER
  PRINT err;errtext$
  PRINT "Error in calculations"
ENDTRAP
```

```
TRAP
  OPEN FILE 2,name$,READ
HANDLER
  PRINT "Disk error!"
  PRINT "Check disk please"
ENDTRAP
```

This page of COMAL information has been brought to you by the good people at:

The Transactor

501 Alden Road, PO Box 3250, Markham Industrial Park, Markham, Ontario L3R 9Z9 Canada

COMAL Benchmarks

COMAL is now running on many different computers. Just for fun, we took our PRIME number SIEVE program, and ran it on every COMAL (even the preliminary ones) that we knew about in North America. The program was run twice, once printing the numbers as they were found, and again without printing them. The results shown below are within a second.

Note: these tests show two comparisons. In some cases, the same COMAL is being run on two different computers. CP/M COMAL runs much faster on the Kaypro than on the C128. In the other case, the same computer is used to run two different versions of COMAL. UniComal IBM PC COMAL was over four times faster than Mytech IBM PC COMAL on our Zenith IBM compatible.

NOT PRINTING NUMBERS (in seconds):

1	Tandy 4000 (80386) UniComal 2.1
8	IBM PC (UniComal 2.1)
13	C64 COMAL 2.0 on C128 FAST
13	C128 COMAL 2.0 FAST
21	MacIntosh COMAL 2.0 prelim
28	C64 COMAL 2.0
28	C128 COMAL 2.0
28	PET 8096 COMAL 2.0 (ROM board)
29	Amiga COMAL 2.0 prelim
31	CP/M COMAL 2.10 on Kaypro
35	CP/M COMAL 2.10 on Epson
38	IBM PC (Mytech 2.0 prelim)
65	Apple COMAL 1.0 prelim
67	C64 COMAL 0.14
72	PET 8032 COMAL 0.14
87	CP/M COMAL 2.10 on C128

Here is the program we used:

```
si#:=3962; count#:=0
DIM flags#(0:si#)
FOR i#:=0 TO si# DO
  IF NOT flags#(i#) THEN
    prime#:=i#+i#+3
    count#:+1
    //print prime#;
    FOR k#:=i#+prime# TO si# STEP prime# DO
      flags#(k#):=TRUE
    ENDFOR k#
  ENDIF
ENDFOR i#
PRINT
PRINT "count=";count#
PRINT "last prime =";prime#
```

PRINTING NUMBERS (in seconds):

2	Tandy 4000 (80386) UniComal 2.1
16	IBM PC (UniComal 2.1)
21	C128 COMAL 2.0 FAST
28	C64 COMAL 2.0 on C128 FAST
38	CP/M COMAL 2.10 on Kaypro
39	PET 8096 COMAL 2.0 (ROM board)
40	C64 COMAL 2.0
43	C128 COMAL 2.0
45	CP/M COMAL 2.10 on Epson
54	IBM PC (Mytech 2.0 prelim)
76	Apple COMAL 1.0 prelim
77	MacIntosh COMAL 2.0 prelim
81	C64 COMAL 0.14
84	PET 8032 COMAL 0.14
111	CP/M COMAL 2.10 on C128
140	Amiga COMAL 2.0 prelim

NOTES

■ Mytech COMAL does not initialize elements in an array automatically as it should. Thus, it needed an extra line to fill the array with 0.

■ A Zenith 151 is our IBM PC compatible.

■ PET COMAL 0.14 did not have enough room to run the full array size. We used a smaller size and estimated what the result would have been with the full array.

■ Thanks to Jeffery Ziebelman at Madison's Radio Shack Computer Center for allowing us to run the program on their new Tandy 4000 computer system.