Commodore 64: Re-Define Your Entire Keyboard With Keywizard

The Transactor®

CDC 00594

May 1985

The Tech/News Journal For Commodore Computers Vol. 5

Issue 06
$2.95

90% Advertising Free!

Programming Aids and Utilities

The Tools For Squeezing The Most From Your Commodore Computer

- STP: Execute Sequential Files
- Exploring The Error Wedge
- Software Numeric Keypad
- List Scrolling Routine
- Directory Gap Remover
- Disk/Extramon 64
- Drive Peeker
- Quote Killer
- BIGPRINT: HI-RES Printer Dump
- A HI-RES Graphics Utility
- Two Short Sprite Editors
- VIC II Chip Parameters



05
0 72246 00594


Transactor
atari
atari
atari

Volume 5
Issue 06
Circulation 63,000

The Transactor

Start Address Editorial 3

Bits and Pieces . 4
C64 IRQ reset
80 Column Right-Justify
C64 Zero Page View
C64 V2 ROM Colour Memory Fix
SYScreeching Off Into Oblivion
Disabling RESTORE on c64
Fast Hi-Res Screen Clear From BASIC
In Search of. . . The perfect colour combination
Put Mental Notes on Disk (or tape)!
Assembler Programming Tip
One Line Decimal to Binary Conversion
The Bleeper
40 Column Wordpro Dump
Regain
Warm Start Border Flasher
Double Width Disk Directory Printout
C64 Easy Disk Status
Bounce 8032
Filename Extensions With SHIFTed SPACE
Easy Screen Print
Phone Speller
Assembler Programming Tip #2
1541/4040 Write Incompatibility Bug
Auto Keywords For The VIC, 64, PET/CBM

Letters 17
Upgrades and Info: DOS 2.7
Clearly Inflexible: Screen Clear Routine
Protection Reflection
Unimplemented Inquiry: 6502 Ops

News BRK 76
New 16-bit Commodores for 1985
Note to Product Review Authors
Over 41,000 Attend World of
Transactor Disk Offer Update
IBM COMAL
The Gold Disk
Quick Data Drive For C64 and VIC-20
Software Developers Newsletter
Commodore Now Provides American
LAMP: Literature Analysis of Microcomputer Publications
Porthole
New Income Tax Program For Commodore PLUS/4
INFOQUICK Bulletin Board for the Commodore 64
The SMART 64 Terminal +4
The FONT FACTORY
CAM-64 (Call Accounting Manager)
Expandable 300/1200 Baud Modem
Printer Ribbons

Introducing VERIFIZER 11
The MANAGER Column 13
TransBASIC Installment #2 19
A New Wedge For The Commodore 64 22
Keywizard For The Commodore 64 26
LINKED LISTS Part 2 31
A HI-RES Graphics Utility 37
VIC Parameters 45
BIGPRINT: HI-RES Printer Dump 48
Two Short Sprite Editors 50
List Scrolling Routine 52
STP: Execute Sequential Files 54
Quote Killer 56
Directory Gap Remover 57
Machine Language Print Loader 58
Aligning The 1541 64
Super Cat 66
Software Numeric Keypad 67
Disk/Extramon 64 68
Drive Peeker 71
File Compare 72

The Transactor
The Tech/News Journal For Commodore Computers

Managing Editor
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Lana Humphries

Contributing Writers
Harold Anderson
Don Bell
Daniel Bingamon
Jim Butterfield
Gary Cobb
John Currie
Elizabeth Deal
Domenic DeFrancesco
Brian Dobbs
Bob Drake
Ted Evers
Mike Forani
Jeff Goebel
Gary Gunderson
David A. Hook
Rick Illes
Scott Johnson
Garry Kiziak
Michael Kwun
Scott Maclean
Allen R. Mulvey
Brian Munshaw
Noel Nyman
Lenard Painchaud
Michael Quigley
Howard Rotenberg
Louis F. Sander
K. Murray Smith
Darren J. Spruyt
Aubrey Stanley
Nick Sullivan
Colin Thompson
Mike Todd

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print" flush right" – would be shown as – print"[space10]flush right"

Cursor Characters For PET / CBM / VIC / 64

Down	– q	Insert	– T
Up	– Q	Delete	– t
Right	–]	Clear Scrn	– S
Left	– [Lft]	Home	– s
RVS	– r	STOP	– e
RVS Off	– R		

Colour Characters For VIC / 64

Black	– P	Orange	– A
White	– e	Brown	– U
Red	– £	Lt. Red	– V
Cyan	– [Cyn]	Grey 1	– W
Purple	– [Pur]	Grey 2	– X
Green	– ↑	Lt. Green	– Y
Blue	– ←	Lt. Blue	– Z
Yellow	– [Yel]	Grey 3	– [Gr3]

Function Keys For VIC / 64

F1	– E	F5	– G
F2	– I	F6	– K
F3	– F	F7	– H
F4	– J	F8	– L

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209, 716-884-0630. **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 876 4741. From Toronto call 826 1662. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ.

Back Issues: $4.50 each. Order all back issues from Milton HQ.

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Volume 4, Issues 04, 05, 06
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 03, 04, 05

Quantity Orders:

MICRON DISTRIBUTING
Micron Distributing
409 Queen Street West
Toronto, Ontario, M5V 2A5
(416) 593 9862
Dealer Inquiries ONLY:
1 800 268 9052
Subscription related inquiries are handled ONLY at Milton HQ

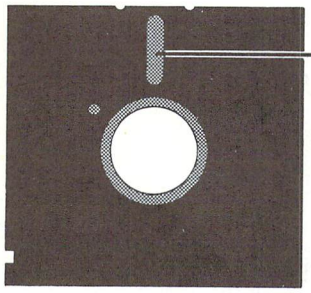
CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 438 8854

U.S.A. Distributor:
Capital distributing
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos of authors or equipment, and illustrations will be included with articles depending on quality. Diskettes, tapes and/or photos will be returned on request.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package.
The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Start Address

Nowhere is the desire to absorb information more prevalent than in the field of microcomputers. Equal maybe, but not more. Of course we all like to learn more about our favourite subject. Just one problem. We live in a mass market society where things are only available for as long as a marketable segment of the population shows a need. Information is no exception. So unless there is a need to continue developing a *subject*, the availability of new information will not only reach a limit, but the need for more will also level off.

The law of "supply and demand" affects all businesses. In the micro industry, manufacturers are responding every year with machines that exceed every specification of its predecessor except price. The information industry barely has a chance to cover all the angles of the current technology before a new line comes along that requires more of the spotlight. Fortunately for us there have not been enough transitions to force discontinued coverage of any one model. However, Commodore is releasing new machines and the story next year will not likely change. Eventually we just won't have room to deal with everything. Recognizing this outcome was one reason behind the theme of this issue and also some slight modifications to our philosophy.

We realize that the majority of computer users have defined an objective for their computer and are either working towards that objective or enjoying the results of having accomplished that objective. And with kind treatment there is no reason why the hardware shouldn't endure indefinitely. In fact, that same computer may never be asked to perform another new duty. Then, when you least expect it, the day will come when change commands priority. Depending how far away that day is, the information that was once so readily available just might be the most difficult to locate. Even the most weathered experts may be unable to offer enough detail to help pinpoint a solution.

But with enough of the right tools, any problem under any circumstances can be eliminated. You've heard the saying, "if you want it done right, do it yourself". Well that's fine if you know how to tackle it, which is usually determined by how much information you can accumulate to provide answers. Thus the "utility" was born – a program that does no favours for the user except to make the process of accumulating information less intimidating.

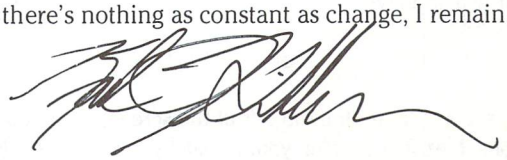
It was around the same time that the "programming aid" became a natural development. Once enough information has been gathered to formulate an idea, the next step is implementation. Programming aids won't provide many *answers*, but unlike utilities their intent is to do all the favours the user expects to make the process of applying all that accumulated information less intimidating.

The philosophy of The Transactor has always been to disseminate information. With this issue, the information we've chosen to disseminate is for the purpose of extracting information as defined by the individual who requires it. We can't possibly supply all the answers, even if we knew all the questions. So the next best alternative is to offer methods to magnify the questions so the answers might be seen more clearly, and to offer a little assistance with the task of implementing the solution. But that's not the most significant benefit.

Learning is a daily process that we all experience, consciously or otherwise. But there is another learning process that too many of us ignore even as we're gathering new facts; the process of learning how to learn. The ability to *systematically* obtain knowledge is what separates those with the skill of total recall from those with the talent for discovering. If you can pioneer a dilemma to its defeat, you have accomplished more than the original problem. Whether you realize it or not, you are acquiring discipline. And with each new accomplishment you acquire a little more. Eventually your battles with new objectives will become merely an exercise in accomplishment. The logical and systematic approach that is so essential in computer science will proliferate into other unrelated functions of your day to day existence. The intimidation of a new frontier will fade faster as you learn to ask why a problem exists and eliminate the problem reason by reason instead of all at once.

The art of problem solving is one that man has been trying to perfect since time began. The concept of computers would never have materialized had there been no struggle with question. Now that we have the computer we must remember that it can only solve problems given enough information, and it's the information we supply that determines how valuable the solutions become. If new solutions do not require new technology then your old computer may never become obsolete.

However, there's nothing as constant as change, I remain,



Karl J.H. Hildon, Managing Editor

Bits and Pieces

C64 IRQ reset

You know the problem: you want to disconnect an IRQ-driven program, but a RESTORE will also reset other things like your screen and border colours. Here's an easy way to set the IRQ vector back to its normal entry point of $EA31:

```
poke 781,12: sys 64701
```

or in assembler:

```
ldx #12
jsr $fcbd
```

80 Column Right-Justify

The ultimate one-liner: when there's a bunch of stuff on the screen of your 8032, enter this:

```
fori = 1to80:?" S ";:forj = 1to24:?" T ":nextj,i
```

(The reverse " T " is an insert). 8000 Series PET/CBM owners. . . try starting the line with:

```
poke 213, 159
```

Quick note: when using the non-relocating load as in: load "file",8,1 you can use any non-zero value instead of 1, so you can use ,8,8 to make typing it in a little easier.

C64 Zero Page View

On the PETs, a good way to get a look at what's going on in zero page was to run an interrupt-driven routine which would continously display the contents of zero page on the screen. Well, on the 64, there's an easier way:

```
poke 53272,7    (,23 to get back to normal)
```

This tells the VIC-II video chip to find screen memory at $0000, giving you a dynamic view of what's going on there. If you have V2 ROMs, you'll have to fill colour memory with something other than background colour to see it, or use the ROM change method below.

C64 V2 ROM Colour Memory Fix

If you have a C64, try this: clear the screen, move the cursor down a line or two, then type:

```
poke 1024, 0
```

If you see an '@' on the top left of the screen, then you have ROM versions 1 or 3. Consider yourself lucky; you can freely POKE to screen memory and see the results of your efforts. If you don't see the '@', then you have ROM version 2. With this ROM, the kernal routine which clears a screen line 'cleverly' fills the corresponding colour memory with the background colour. Since background equals foreground the result is a truly clear screen. Furthermore, if you've ever run a program for the 64 or typed in a little screen blitz from a magazine that didn't work, it could be because the author wrote it on a V1 or V3 machine and assumed it would work on any 64.

The solution? If you're willing to forsake the RAM underlying the kernal ROM for this cause, you can correct the foolish behaviour by changing just two bytes. First copy the BASIC and Kernal ROM into the underlying RAM. If you have a Machine Language Monitor with a 'Transfer' command (like Supermon or Micromon), this can be done with these two operations:

```
t a000 bfff a000
t e000 ffff e000
```

This transfers the contents of the 8K BASIC ROM and the 8K Kernal ROM into RAM. But it gets it *from* ROM. . . why does it not try to put it back in ROM? Because the 64 knows you can't possibly mean that thanks to a chip called a FPLA (Field Programmable Logic Array). This redirects data flow to a logical destination that has been preset by the engineers. And yes, it's fast!

Next switch out the ROM and switch in the RAM by putting a 53 decimal into the bank select register (location 1):

```
BASIC:   poke1,53
Monitor: : 0001 35
```

Now, at $E4DA, there is the instruction:

```
                LDA $D021
Change this to: LDA $0286
like this:      : e4db 86 02
```

($0286 holds the current cursor colour)

The kernal will be living out of RAM from now on, but POKEing to the screen will always yield visible characters. Seems like a lot of work for just poking to the screen, especially when you could have merely changed the background colour. But there was another reason for this excercise (of course).

Now that you have all of your 64 operating from RAM, anything can be changed. The spelling of keywords and error messages are fun to modify, but more importantly the ROM routines can be altered. JMP instructions can be re-routed or entire routines can be substituted. Most common is the "BRK instruction insert" for examining the state of the machine at any particular point in a routine. With your favourite dissassembler you simply change the first instruction beyond the last instruction you want executed to a BRK ($00). Now when you cause that particular stretch of code to execute it will stop at the BRK and you can peer around awhile.

Logically you should be able to replace the BRK with the instruction you wiped out and continue executing. Some routines will allow such interruptions but others aren't so tolerant. Most likely you'll need to replace the BRK and start over (perhaps with a BRK somewhere else?).

SYScreeching Off Into Oblivion

On any BASIC 4.0 machine, you can easily enter the monitor with SYS4, right? Well, try it with a quote after the 4 like:

```
SYS 4"
```

What happened? We won't spoil it by giving it away – look up the purpose of location 4 to figure it out.

Disabling RESTORE on c64

If you don't want someone crashing out of your program with the RUNSTOP/RESTORE sequence, here's an easy way to disable it:

```
poke 792, 193   (,71 gets back to normal)
```

The disable POKE pretty much renders the NMI routine impotent, so RS-232 operations won't work while it's in effect.

Quick Note: 255–x = 256+not(x)

Fast Hi-Res Screen Clear From BASIC

Last issue's Bit's & Pieces gave a little machine language routine to quickly clear bit-mapped memory. Since then Nick Sullivan from TPUG magazine showed us this neat trick to accomplish the same thing from BASIC. If you create a large array and then CLR it, BASIC will zero out anything in its path, including hi-res screen memory if it happens to be in the way. If you have a hi-res screen within the limits of BASIC variable space, just put this line at the beginning of your program:

```
clr: f=fre(0):dim a((-65536*(f<0)+f)/5-10): clr
```

That's it! Within a second, the screen will clear. You can't use this trick if your screen memory is at $C000, but at the usual spot at $2000, and with BASIC pointers set up normally, it works like a charm.

In Search of. . . The perfect colour combination

Looking for the perfect background/border/character colours for programming on the C64 with a 1701/1702 monitor? Try this:

```
poke 53281,0 : poke 53280,11 (press Commodore-2)
```

For the VIC:

```
poke 36879,9 (press CTRL-8)
```

Adjusting the bright/contrast controls to look good with this combination results in an easy-to-look-at screen for hours of programming without fried retinas.

Quick Note: If processing time is critical, you can speed up the CPU by turning off the VIC-II video chip in the c64: poke 53265,peek(53265) and 239.

Put Mental Notes on Disk (or tape)!

Ever compose your thoughts idly on the screen of your computer? Or draw a neat picture using graphics symbols while idly talking on the phone? Want to save the screen to disk or tape to bring it in again later? Enough questions, here's what to do. Last issue's Bits & Pieces gave a method to save a range of memory. To save the screen (at $0400 on the C64):

```
sys57812"filename",8:poke193,0:poke194,4:
poke174,231:poke175,7:sys62954
                              (use ',1,1' for tape)
```

Of course, that'll mess up a bit of the screen: that's the catch. To bring back your screen, just LOAD it any time with:

```
load"filename",8,1  or load"filename",1,1 for tape
```

With a BASIC 4.0 machine, just use the monitor to save the screen:

```
sys4
s"filename",08,8000,83e7
(,8000,87cf for 80 column machines)
```

Unfortunately you can't save memory above $8000 to tape. Pardon me. . . you can save it to tape, you just can't LOAD it back. Commodore never expected anyone to require memory above $8000 to be saved so they used the high bit of the address for something else. In the VIC or 64 this anomoly has been dealt with and whatever that bit does is now seperated into its own byte.

Assembler Programming Tip

Branch instructions like BNE, BEQ, BPL, etc. can be a pain when your program grows and the branch can't reach the intended destination any more – the assembler gives a "BRANCH OUT OF RANGE" error. You can get around this problem by branching to a JMP somewhere, but for a short easy way to do long branches, consider this:

```
intended branch : BNE SOMPLC
easy long branch: BEQ * + 5: JMP SOMPLC
```

This leaves the intent of the branch clear, and doesn't force you to define a meaningless label somewhere.

One Line Decimal to Binary Conversion

Store the value (0–255) to be converted in 'x', then:

```
z$="":forj=0to7:k=x/2:x=int(k)
:z$=mid$(str$(k<>x),2)+z$:nextj:printz$
```

The Bleeper

This little noisemaker runs on any PET with CB2 sound:

```
10 poke59467,16:fora=1to255:poke59466,a
   :forb=1to255stepa:poke59464,b:nextb,a
20 print chr$(7)
```

40 Column Wordpro Dump

Here's a small program that will print out a Wordpro-format text file to the screen. It will work with Paperclip, but there will be a few bytes of garbage printed at the beginning as Paperclip stores extra information at the start of its files.

```
10 rem* print a wordpro file to screen
20 rem* 40 column version for 4032/c64
100 input "filename";f$
110 open1,8,0,"0:"+f$
120 b=1984
130 rem b=33728 for 4032
150 cc=peek(646): c=54272: rem* only for 64
160 print "qqqqqqqqqqqqqqqqqqqqqqqqq"
165 rem 25 cursor downs
170 get#1,a$,a$
175 rem- main loop -
180 for i=b to b+39
190 get#1,a$: poke i,asc(a$+chr$(0))
210 poke c+i,cc: rem* only for 64
220 if st then 250
230 next i: print: goto180
240 :
250 close1:end
```

Regain — Lenard Painchaud

```
5 rem* restore pgm after reset or new *
10 ad=49152:fori=0to21
20 readd:pokead+i,d:nexti
30 data 169,8,141,2,8,32,51,165,24
40 data 165,34,105,2,133,45,165,35
50 data 105,0,133,46,96
60 print "to execute this program, use:"
70 print " sys";ad;":clr"
```

Lenard writes: "It comes in handy when a program crashes and you can't get your cursor back. Before you can use this program, however, you need a reset switch. When you turn on your computer, load and run the regain program. Now, when the computer crashes, press the reset switch. That doesn't do the trick though. You then have to type sys 49152. Now you will have your program back. You can change the memory location where the ML program is stored by changing the value of AD in line 10."

Note: To reset your computer, you have to momentarily ground pins 3 on the user port – pin 1 is a ground. Connecting a push button across pins 1 and 3 makes a good reset switch – It can save your program's life! The above program will also bring back a program after a NEW.

Warm Start Border Flasher — Nick Barrowman, St. John's, NFLD.

Nick writes: "This small routine doesn't serve any practical purpose but it is an example of how you can use the main basic program loop vector in the C64 (warm start link at $0302). A more practical purpose is auto-run routines. This routine will change the colour of the screen border whenever <return> is pressed (from BASIC) or when a break or restore is performed. Hope you like it!"

```
10 fora=49152to49169:readb:pokea,b:c=c+b:nexta
20 ifc<>1779thenprint "checksum error!":stop
30 sys49152
40 print "basic warm start flasher activated"
50 data 169,11,141,2,3,169,192,141,3
60 data 3,96,238,32,208,76,131,164,0
```

Double Width Disk Directory Printout — Brian Dobbs

The following little program will give you a disk directory in two columns, useful for printing out and putting in the disk sleeve. If sending the directory to the screen, it will appear as a normal directory on a 40 column screen, and double width on an 80 column screen.

```
100 rem** directory double width **
110 rem**    by brian dobbs    **
120 rem**   timmins, ontario   **
130 k=4: rem* k=3 for screen, 4 for printer *
135 r=1: open k,k
140 dr=0: rem* directory drive zero *
150 gosub 220: rem* directory subroutine
160 close3
170 input "another (y/n)";an$
180 if an$<>"y" then end
190 print "insert another disk, press any key"
200 geta$:ifa$<>"" then200
210 goto130
220 n$=chr$(0):h=256:open1,8,0,"$"+mid$(str$(dr),2)
230 get#1,a$,a$
240 get#1,a$,a$,a$,a1$: if st then 290
250 d=asc(a$+n$)+asc(a1$+n$)*h: print#k,d;
260 get#1,a$:ifa$<>"" then print#k,a$;:goto260
270 r=r+1:ifr=2thenr=0:print#k :goto240
280 d$=str$(d):print#k,tab(40);:goto240
290 close1
300 return
```

C64 Easy Disk Status — John Currie, Mississauga Ont.

This tidy little routine sits in the cassette buffer at 828, and will display the current disk error status when executed. It's very handy, since the C64 has no built-in disk status function.

```
100 rem basic loader for disk status
110 a=828
120 read b:c=c+b:if b=256then140
130 poke a,b:a=a+1:goto120
140 if c<>8574then print "error in data stateements": end
150 print "'sys 828' returns the current disk status"
160 data 169,  0,  32, 189, 255, 169,  15, 162
170 data   8, 160,  15,  32, 186, 255,  32, 192
180 data 255, 162,  15,  32, 198, 255, 169,   0
190 data 141,  19,   3,  32, 228, 255, 172,  19
200 data   3, 238,  19,   3, 153, 127,   3, 201
210 data  13, 208, 240,  32, 204, 255, 169,  15
220 data  32, 195, 255, 160,   0, 185, 127,   3
230 data 170, 200,  32, 210, 255, 224,  13, 208
240 data 244,  96,   0, 256
```

Bounce 8032

Here's another one of those useless little special effects. For some reason though, this one can hold your attention for hours (well, minutes maybe). It only runs on 8032's, since it uses the scroll down feature unique to that machine.

```
5 sp=32768:forj=0to1step0:s=153-128*k:k=1-k
10 fori=1tornd(1)*15:printchr$(s);
   :pokesp+rnd(1)*1000,46:nexti,j
```

Filename Extensions With SHIFTed SPACE

Filename extensions such as .SEQ, .ASM, .OBJ, etc. are useful to indicate file types, but some programmers prefer to use a shifted space instead of a period in the filename. Such a file will be listed in the directory with the extension OUTSIDE the quotes around the filename. To load the file back in, you can specify the filename without the extension, or specify the entire filename (including the shifted space) if greater uniqueness is required. You can also use this method to make "notes" about a file — the note will show up in the directory but need not be entered to load the file in.

Easy Screen Print

A powerful and little-used feature of Commodore BASIC is the ability to use a screen file for INPUT. If you open a screen file and then GET or INPUT from that file, you will read characters directly from the screen starting at the cursor position, and advance the cursor to the next character or INPUT field.

There are all kinds of uses for screen input, but a good application is to convert screen memory character codes to their CBM ASCII equivalents. Such conversions are necessary when printing all text on the screen to a printer. The following line of code will dump an 8032's screen to a Commodore printer with an 80 column margin width.

```
1 open3,3:open4,4:print"[S]";:fori=1to80:get#3,a$
  :print#4,a$;:next:close3:close4
```

For 40 column machines or a printer set for column widths greater than 80, use this version — it prints a carriage return every 40 characters:

```
1 open3,3:open4,4:printchr$(19);:fori=1to24
2 forj=1to40:get#3,a$:print#4,a$;:next j:print#4," ":next i:
close3:close4
```

Phone Speller

Some telephone numbers are most easily remembered by the letters on the dial. For example, you can get information on 1985 Volkswagens by calling 1-800-85-VOLKS. Wouldn't it be nice to give your friends a similarly catchy way to remember your number? The following program (it works on any machine) gives all letter combinations from any phone number (zero and one have no associated letters, so 0 or 1 appears). There are 2,187 combinations for a 7 digit number, so be prepared for a long list. And even if there are no pronounceable words in the list, you can invent acronymns. What better way to spend an afternoon than to find phrases to fit 2,187 acronyms?

```
100 rem* phone speller *
110 rem* dec84/cz     *
120 :
130 open1,3 :rem 1,4 for printer
140 l$="000111abcdefghijklmnoprstuvwxy"
150 :
160 input"phone number";pn$
170 n=len(pn$)
180 dim p(n), n$(n)
190 :
200 for i=1 to n
210 n$(i)=mid$(l$,val(mid$(pn$,i,1))*3+1 ,3):p(i)=1
220 next i
230 rem* n$ holds letter groups for each digit in number *
240 :
250 for i=1 to 3↑n
260 print#1,i,
270 for c=1 to n:print#1,mid$(n$(c),p(c),1);:next c
    : print#1,chr$(13);
280 carry=1
290 for j=1 to n
300 p(j)=(p(j)+carry): carry=0
310 if p(j)>3 then carry=1:p(j)=1
320 next j,i
```

Assembler Programming Tip #2

If you've ever looked through someone's machine language program and come across a seemingly useless BIT instruction (eg. BIT $FFA2), or an inexplicable .BYTE $2C, there is a method to his madness.

The BIT instruction doesn't do any harm to memory or CPU registers, it just sets the zero, minus, and overflow flags based on the contents of the given memory location. In some instances, BIT is used almost like a NOP, but with one major difference: the two operand bytes used to specify the memory location are part of the instruction, and so are not executed as instructions if the BIT is executed. If the first byte of the instruction ($2C) is skipped however, you can execute a 2-byte instruction. For example, consider the following assembler code:

```
ENTRY1  .BYTE $2C
ENTRY2  LDX #$FF
```

If a program were to execute the code starting at ENTRY1, the CPU would see a $2C which is a BIT instruction, and interpret the next two bytes (the LDX instruction) as the argument for the BIT — in this case, the CPU would see:

```
BIT $FFA2
```

If the $2C was skipped over and instructions were executed from ENTRY2, the CPU sees the bytes $A2, $FF and interprets the LDX #$FF instruction normally.

Using the above technique allows you to enter a routine with the X register intact, and later enter the routine one byte past the start and have the register changed to something else before the routine does its thing. Of course, any register may be used instead, or any 1 or 2 byte op code can be executed after the $2C.

The technique is explained here in case you come across it in someone else's program, since it's a fairly widely used and accepted 6502 programing practice. Generally though, programmers who use tricks like this enjoy writing obscure code to save a byte or two of

memory, and don't care if anyone else can look at the program and understand it. Many programs, including those printed in the Transactor, are designed to be easily read by people, not computers, and should keep away from such brain-twisting exercises. But giving such advice to a hacker is about as effective as advising a kid not to step in puddles on his way home from school.

1541/4040 Write Incompatibility Bug

When the 1541 single disk drive arrived, so did a new buzz word: "write-compatible". At first it seemed that diskettes were completely portable between 4040 and 1541 drives. Then reports of some nasty disk failures started circulating. Here's why.

Every sector on a disk starts with a "synchronizing character", a Header block, another sync character, and then the data stored in that sector. "Physically" it looks something like:

(.... = sync **HHH** = Header **DDD** = Data)

4040: **.....HHHHH.....DDDDDDDDDDDDDD.....**
1541: **.....HHHHH...DDDDDDDDDDDDDD.....**

Notice how the second sync on a 1541 disk is shorter than on the 4040. Now you take a 4040 disk and write on it with a 1541. It becomes:

.....HHHHH...ddddddddddddddddddd.....

But that's OK - the 1541 and the 4040 can still cope. There is still enough of the sync and the data block is still the same "length". However, go back to the 4040 and write to the same sector and:

.....HHHHH...dddDDDDDDDDDDDDD.....

Blammo! The data block starts with residue data from the 1541 write to the second sync character. The data block is now "too long" and the disk returns Read Error 23: Checksum Error in Data Block

Apparently new 1541's (as of July 84) have been modified to allow write compatibility between all 1541 and 4040 diskettes.

Auto Keywords For The VIC, C64, PET, and CBM

Today we have the contender for the 'two liner' of the year contest. This machine language monster consumes less than the equivalent of two lines of BASIC. It sits in the cassette buffer and will reconfigure every (shifted) letter on your keyword to produce a keyword. It's IRQ driven, but retains the old IRQ to jump through at the end, so if IRQ driven code is already installed, this program won't bother it. The code also operates in direct mode only, which most can appreciate if INPUT statements are used in your program. And, if it comes down to it, the "\" key on the PET/CBM or the (shifted) pound symbol on the C64/VIC will reset the original IRQ and kill the routine.

Now, considering that there are only 26 letters on the keyboard, how are all the keywords accessed? With the VIC and C64 we have 76 keywords in total, and with the PET/CBM models with BASIC 4.0 we have 91. To battle this problem, a memory location within the routine can be altered to supply you with every keyword. This location defines a "window" over the total set of keywords. You can't get access to all the keywords simultaneously, but you can move the 26 keyword window over any part of the command set (ie. the part you use most). Check out any list of keywords for your optimal window.

As shown, the program will give you the first 26 keywords. Since the first keyword is "END", a shifted-A will print "END". Vary location 683 from 128 to 193 for the PET/CBM, or location 882 from 143 to 193 for the VIC and C64. Lower values will move the window over the error messages.

Note For PET/CBM Users: Reset IRQ before LOADing from disk, then sys(634) to start again. The C64 and VIC do not have this bug, but the PETs sure do. The machine will hang until the STOP key is pressed if any IRQ driven wonder is present during a LOAD.

```
10 rem save "0:keyword pet.bas",8
100 rem ** rte/84 - auto keyword for the pet/cbm
110 for j=634 to 774: read x: poke j,x: ch=ch+x: next
120 if ch<>17758 then print "checksum error": end
130 print "sys(634): rem ** to enable": end
140 data 165, 145, 201,   2, 240,  20, 165, 144
150 data 141,   5,   3, 165, 145, 141,   6,   3
160 data 120, 169, 149, 133, 144, 169,   2, 133
170 data 145,  88,  96, 165,  55, 201, 255, 208
180 data  90, 165, 217, 201,  92, 240,  87, 201
190 data 193,  48,  80, 201, 219,  16,  76,  56
200 data 233, 193, 170, 169,  27,  32, 210, 255
210 data 169, 157,  32, 210, 255, 169, 178, 133
220 data  87, 169, 176, 133,  88, 160,   0, 132
230 data  89, 224,   0, 240,  21, 177,  87,  24
240 data  42, 176,   8, 200, 208, 247, 230,  88
250 data  76, 199,   2, 200, 230,  89, 228,  89
260 data 208, 235, 177,  87, 133,  90,  36,  90
270 data  48,  11,  32, 210, 255, 200, 208, 242
280 data 230,  88,  76, 220,   2,  56, 233, 128
290 data  32, 210, 255, 108,   5,   3, 173,   5
300 data   3, 133, 144, 173,   6,   3, 133, 145
310 data 108,   5,   3,   0,   0
```

```
10 rem save "0:keyword c64.bas",8
100 rem ** rte/84 - auto keyword for the commodore 64
110 for j=828 to 970: read x: poke j,x: ch=ch+x: next
120 if ch<>17162 then print "checksum error": end
130 print "sys(828): rem ** to enable": end
140 data 173,  21,   3, 201,   3, 240,  24, 173
150 data  20,   3, 141, 201,   3, 173,  21,   3
160 data 141, 202,   3, 120, 169,  92, 141,  20
170 data   3, 169,   3, 141,  21,   3,  88,  96
180 data 165,  58, 201, 255, 208,  85, 165, 215
190 data 201, 169, 240,  82, 201, 193,  48,  75
200 data 201, 219,  16,  71,  56, 233, 193, 170
210 data 169,  20,  32, 210, 255, 169, 158, 133
220 data  87, 169, 160, 133,  88, 160,   0, 132
230 data  89, 224,   0, 240,  21, 177,  87,  24
240 data  42, 176,   8, 200, 208, 247, 230,  88
250 data  76, 137,   3, 200, 230,  89, 228,  89
260 data 208, 235, 177,  87, 133,  90,  36,  90
270 data  48,  11,  32, 210, 255, 200, 208, 242
280 data 230,  88,  76, 158,   3,  56, 233, 128
290 data  32, 210, 255, 108, 201,   3, 173, 201
300 data   3, 141,  20,   3, 173, 202,   3, 141
310 data  21,   3, 108, 201,   3,   0,   0
```

VIC users need only make one change. The number 160 in bold becomes a 192 (Also add 32 to the checksum just for completeness)

Letters

Upgrades and Info: 1) I have seen references to DOS 2.7 for the 4040. How does this differ from other DOS's? In particular, is there any point in me taking my old 2040 drive which has been upgraded with ROMs and new 6530 to DOS 2, and burning a set of ROMs (or even purchasing them, if they exist) to give me an upgraded–again disk drive? For instance, one of the things that I dislike about my present DOS 2 is that LOAD "filename",8 starts both drives going, and waits until the wrong drive doesn't find the program before it sends out the program it did find on the right drive. So the questions are: Is there really a DOS 2.7 for the 4040? If so, can I upgrade? What's involved: new ROMs? New 6530? Minor or major board surgery? How can I get any chips or the code to burn my own?

2) A lot of people locally seem to be getting the B–128 package from Protecto. I am already getting calls about do I know this or that about it. I have heard that Commodore, while no longer producing B–128's, has not finished either. Do you have any idea of what may be available about the B–series machines in other countries, say England or Germany, or contacts from whom I could find out? I would dearly like a PRM and schematics; my German and Swedish are fluent enough so that information in those languages is perfectly fine for me, and even French would be OK in the absence of anything else.

Charles McCarthy, 1359 W. Idaho Avenue, St. Paul MN 55108

Pretty terrific questions, with some not so terrific answers. First, the DOS question.

As far as we know, DOS 2.7 is currently available for the 8250 and 8050 disk drives only. The problem with the DOS going to the currently unused or wrong drive first is still a problem. It even seems worse with the 8250. Instead of politely determining that an empty drive is not in use, the DOS fires a couple of BUMPs onto the job que in (what seems) a last–ditch effort to avoid the occupied drive. Once all the noise and vibration dies down, it goes to the correct drive. Once this routine has been suffered once, the DOS is bright enough to realise there is no disk inserted, and won't try it again.

One major improvement with DOS 2.7 makes this bearable though – Relative file length has been extended to the maximum capacity of a diskette (8050 or 8250). The old limit was 180K. Since 4040 drives only hold 170K, even if an upgrade kit is available, this improvement makes a total upgrade somewhat less than worthwhile.

Only one other noticeable difference worth mentioning – An extended error message is also generated, with one extra digit at the end to inform you of what drive has been generating the message. This is alright at times, but is really aggravating when it comes time to use a program such as Petspeed. Petspeed errs out due to the length of the error message. The extra length means an error to Petspeed, so the program always gets upset and resets. No fun. Otherwise, DOS 2.7 is a pleasure, and might be worth your time if you have an 8050 and need longer REL files.

Next, on to question number two. The B series package has sure developed a lot of activity in North America, if the volume of mail and the calls we're getting is any indication. $900 US for the B machine, 8050 drive, 4023 printer, green phosphor monitor, and a few diskettes to boot, who could go wrong. You could literally buy the system for the drive alone, and have all these extra goodies as a bonus. One problem. Very little documentation available.

In Canada, Commodore was pretty good to the software developers before the computer was officially released. Everyone of any importance got a B machine sent to them for software development. Some of the better packages were converted, but were shelved due to the non–release of the machine in Canada. Now, with Commodore releasing their entire stock of the B on Protecto, plus some more extra stock to make it more attractive, the B is alive again, but only in the USA. We know the machine to be pretty good, once you get used to the bank switching and playing about with the stack, etc. But there is really little available about it. We published Jim Butterfield's B Series ROM maps back in our reference issue in November 1983, and we are again publishing the maps in our reference book (The Complete Commodore Inner Space Anthology), but that seems to be the extent of high level info. A few magazine articles have appeared in TORPET, TPUG magazine, Compute!, and of course, The Transactor, but again, it doesn't really add up to a hill of paper.

If anyone reading has some special knowledge on the subject, from any corner of the globe, please send it in to us. For all the programmers here in good old Canada, maybe it's time to retrieve a few wasted dollars by releasing your creations. Let us know, and we'll pass it on. To start the ball rolling, here's a tip from Dave Berman, of Weston, Ontario. Place a (rvs) 'c' within quotes in a REM line in your BASIC program, and your program will automatically DLOAD and RUN the first program on disk when listed to the screen – one way to discourage code peekers. That's the first B bit, your letters will supply us with more.

Clearly Inflexible: I have written a good program in which I have a heading that must remain on the top 3 lines of the screen at all times.

This heading is used to print the different functions of the program, just like a word processor. These functions are written by poking the letter's screen display codes in the screen memory (from 1024 to 1144) for the 3 lines.

The problem is that in the program, at some times, I have to clear the screen without erasing the heading. The only way that I have found is poking the value of the space character (chr$(32)) in each of the screen addresses remaining. ie. from 1045 to 2023. However, this is awfully slow. Is there a way to accelerate that function. I have looked in a few books to find a way to control the clear screen, but without any solution.

There are a number of ways that this task can be completed. The first is to write a simple IRQ controlled routine to constantly refresh the top three lines from a data location somewhere else in memory. All that would be required of you is to POKE the correct display lines into this memory address, and the IRQ would transfer the display for you. If a clear screen is executed, the IRQ would only allow the screen to remain clear for a maximum of 1/60 of a second. A clean way to cure the problem.

A second method is to write a simple clear screen routine in assembler, that starts at the fourth line on the screen. When a clear screen is required, SYS to the start address of the routine and let it do a partial clean.

For you, the routines below have been prepared. Each are written in PAL format, but the op codes are standard MOS syntax, so any mini-assembler would probably work for you.

```
100 ;** irq driven display routine **
105 ;
110 * = 828      ;start addr can be anywhere convenient
115 ;
120 irqvec = $0314          ;irq vector in ram
125 screen = 1024           ;start of screen
130 ;
135 sei
140 lda irqvec : sta oldirq   ;retain old irq vector
145 lda irqvec + 1 : sta oldirq + 1
150 lda #<start: sta irqvec   ;point new irq vector at code
155 lda #>start: sta irqvec + 1
160 cli: rts
165 ;
170 start = *
175 ;
180 ldx #0
185 ;
190 loop = *
200 ;
205 lda data,x              ;get the display data
210 sta screen,x            ;store it on the screen
215 inx: cpx #120           ;only allow 3 lines (3 x 40 = 120)
220 bne loop
225 jmp (oldirq)
230 ;
235 oldirq .wor 0           ;two bytes storage for old irq vector
240 ;
245 data = *                ;store the display lines from here on
250 ;
255 .end
```

```
300 ;** routine to clear all but the top 3 lines of the screen **
305 ;
310 * = 828       ;start addr can be anywhere convenient
315 ;
320 screen = 1144           ;new screen start address for clear
325 ;
330 ldx #0: lda #32         ;(space)
335 ;
340 loop = *                ;loop to clear 3 pages of screen mem-
                            ory past the top 3 lines
345 ;
350 sta screen,x : sta screen + 256,x : sta screen + 512,x
365 inx : bne loop
370 ;
375 final = *
380 ;
385 sta screen + 768,x      ;final clear of the bottom
390 inx: cpx #112           ;balance of screen to clear
395 bne final
400 rts
405 ;
410 .end
```

Protection Reflection: In response to your Transactor issue dealing with the implementation of various types of protection schemes, one particular form that I have recently heard of was unmentioned. This has to do with the accessibility of more than the standard 1541 35 track capabilities. I have been told by a few people that the 1541 along with the C64 is capable of formatting and using more than 35 tracks, but that Commodore didn't let anyone in on the big secret of how to go about this.

I am very curious about this claimed ability and would very much like to know how this can (or can't) be done. Also, could you possibly tell me where I could find out other little secrets about the 1541 (like the ones you mention in your Nov. issue) or is the only way to learn these by experimentation? Ryan Briegal, M.S.U., E. Lansing, MI.

The 1541, along with the 2031 and 4040 drives, are capable of moving their stepper motors in half steps, thus increasing their capacity to 70 tracks, in relation to the normal 35. However, they cannot be expected to accurately read and write in this manner. At times it would work, at others it might not. So what at first might seem like a design oversight is in fact quite intentional. To get higher capacity with reliability, you need more accurate motors which cost more money.

To date, numerous protection schemes have been designed that make full use of the half step capability of the drive. A portion of the diskette is formatted in this rather obtuse manner, either a half step out from a normal diskette, thus writing and reading in a special way, or the balance of the diskette is half stepped right across, and data is stored between the normal tracks. As long as the data stored in between is not required for anything more than a check of protection, and the program realizes that it may take a few tries to get the info, then the technique is ok. It's your choice if you want to start drive stepping, but get ready for some pretty heavy duty programming.

For more facts and info, there is a really good book on the subject. The name is 'Inside Commodore DOS', written by Richard Immers and Gerald G. Neufeld, published by: Datamost, 20660 Nordhoff Street, Chatsworth, CA, 91311-6152 (818) 709-1202, ISBN 0-88190-366-3. For me, this book is to the drive as Raeto Collin West's book is to the computer: a pure and applied source knowledge.

Unimplemented Inquiry: In the Transactor of January 1985 (pg.22), I was surprised to learn that most 65xx CPUs have operations that can be executed but are not officially part of the instruction set. Where would I find more information (definition and machine code)?
Richard Pitre, Montreal, Quebec

For some really good info on the pseudo-ops that MOS won't talk about, there are two pretty good sources. Raeto Collin West's 'Programming The PET/CBM' has an in depth article on the subject on pages 488 and 489. In his usual thorough style, Mr. West lists the codes, their functions, and a few paragraphs relating some experiences. To further compliment your quest for knowledge, you could try the October 1983 Issue (Issue 41) of Compute! Magazine, on pages 261 through 264. The article 'Extra Instructions', written by Joel C. Shepherd, has been written with a pro-use attitude. However, there are those that have less than total faith in the extra codes (Jim Butterfield for one). Further, MOS knows they exist, but neglect to document them. From a market standpoint it would have been to their advantage to include the extra instructions in their command set, but they didn't trust them enough to do so. Heed these words of warning before ever considering writing code with the unofficial pseudo-ops in place. If the people who designed the chips don't trust them, why should you?

Introducing "VERIFIZER"

The Transactor's New Foolproof Program Entry Method

The greatest source of calls and letters we get at the Transactor are from people who are having difficulty getting a long program from the magazine to work right. Occasionally it's due to a goof-up on our part, but other times it's simply because the chances of making a typing error in a very long program are quite large, and finding that error (probably more than one) after the entire program has been entered is quite difficult. This is especially true for machine language programs in BASIC loader form, where the hundreds of numbers in the DATA statements can make the most competent reader go cross-eyed.

To save you from endless program de-glitching sessions, we've come up with two solutions. The first is the best, but it'll cost you: All programs printed in the magazine are also available on disk, including those from past issues (see the center insert in this issue for more details).

The second solution doesn't save you any typing, but it can help you find errors as soon as you make them. The solution comes in the form of a short machine language program called "VERIFIZER" — so named, of course, because it verifies! (OK, so we indulged in a cute name — no nasty letters from English teachers, please.) VERIFIZER should be run before typing in any long program from the pages of The Transactor. From this issue onwards, all long programs will have two uppercase letters printed before each line. The programs should be entered normally (i.e. don't enter the letters in front of the line), but every time you enter a line, VERIFIZER will print two characters at the top left of the screen. If these letters match those in print, you entered the line properly.

Besides catching omissions and other blatant errors in each program line, VERIFIZER will also catch transpositions; probably the most commonly made error with numbers in PEEK, POKE, or DATA statements. For example, if you're typing quickly and enter 52381 instead of 53281, VERIFIZER will catch the error and report a code different from the one printed with the program listing. To make your life easier though, there are some variances that VERIFIZER will NOT catch. One is that it ignores spaces — you can put as many spaces as you want between keywords, or leave them out altogether without changing the reported code. Spaces are accepted anywhere as long as they don't split up keywords. Keep in mind, though, that there may be occasions where the number of spaces is important (eg. in a string definition), so take notice of messages such as [9 spaces] in a listing - which means, naturally, enter 9 spaces, not the number 9 followed by the word 'spaces'. Another thing you can get away with is the abbreviation of keywords. Since VERIFIZER looks at token values, you will get the same result if you type 'nE' or 'next'.

How to use VERIFIZER

Listing 1 contains two versions of the VERIFIZER program: one for the PET, and one for the C64 or VIC. These listings will appear in all issues from now on. You'll notice that the verifizer listings are themselves printed in "verifized" format (with the report codes along the left side), which seems pretty silly, since you won't have verifizer while you're typing the program in. The codes are there just as a check, so that once verifizer is up and running you can press RETURN over some program lines and see if the reported codes match up with those printed with the listing.

Enter the applicable program and RUN it. If you get the message, "***** data error *****", re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, enter NEW, then turn VERIFIZER on with:

SYS 828 to enable the C64/VIC version (turn it off with SYS 831)
or SYS 634 to enable the PET version (turn it off with SYS 637)

Once VERIFIZER is on, every time you press RETURN the two-letter report code will appear in the first two screen locations in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC). If you press RETURN on a screen line that doesn't contain a BASIC program line (i.e. no line number at the beginning), the code you'll get is unpredictable — VERIFIZER is only used to report on BASIC program lines.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. If you're in the habit of re-numbering segments of a program as you type it in, be prepared for unmatched codes, since VERIFIZER uses the line number as part of its checksum calculation. Once the program has been properly entered, be sure to turn VERIFIZER off with the sys indicated above before you do anything else.

VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

How VERIFIZER Verifies

VERIFIZER generates the report code on a checksum principal, but assigns "weights" to each byte to catch transposition errors. The weights 1,2,3,4 are assigned to the tokenised BASIC line, and the low byte of the line number is added in. In other words, a checksum is obtained by taking the line number low byte, adding 1 times the first byte in the BASIC line (from the BASIC input buffer at $0200), 2 times the second, 3 times the third, 4 times the fourth, then 1 times the fifth, etc. The final checksum is a two byte unsigned integer. The high byte is discarded, and the low and high nybbles of the low byte are used to form the two report code characters, which are put directly into screen memory in the PET version, and PRINTed to the screen with the VIC/C64 version.

Since only the low byte is used, checksum variances of exact multiples of 256 will not be detected, but the probability of that occurring as a legitimate error was judged to be reasonably small. For example, if you "accidentally" entered the line number 1256 instead of 1000, there would be no discrepancy in the report codes. However, if you're that uncoordinated, don't even try to type in programs — get the disk.

VERIGEN: VERIFIZER program generator

On the other side of the VERIFIZER story, there is VERIGEN to create a verified sequential file from a program. The version we use includes special control characters in the output file for the typesetter, but the version shown here (listing 2) just creates a sequential file giving the program listing with line numbers separated from the verifizer codes by a single space.

What good is VERIGEN to you? Well, if you have a printer and like storing program listings as a final backup, you may want to have them in verified form should the day ever come when every one of your disks instantly turns to electro-dust. Or if you have a 64 with a 1541 drive and want to give a program to a PET owner who has an 8050, you can give him a verified listing (assuming, of course, that he has a copy of VERIFIZER).

How to use VERIGEN

VERIGEN runs on a C64 and works with two disk files, "infil" and "outfil". Before executing Verigen, the program to be verified must be in memory, AND a listing of that program must be on disk as a sequential file named "infil". To create infil, just enter the following after the input program for Verigen is in memory:

```
open8,8,8,"@0:infil,s,w": cmd8: list
print#8: close1
```

After "infil" has been set up in this way, just sys49152 to execute Verigen. It will create "outfil" for you, which will be a sequential disk file containing the program listing with verifizer codes before the line numbers. Outfil can be sent to a printer or manipulated like any sequential text file (i.e. loaded into a word processor).

VERIFIZER will make pretty sure that the program you type in mimics that in the listing, but it can't guarantee that the listed program is bug-free. So if VERIFIZER says a program is OK and it still doesn't work, you can drop us a line without worrying that the problem may be with your typing.

Listing 1a: VERIFIZER for C64 and VIC20

```
KE 10 rem* data loader for "verifizer" *
JF 15 rem vic/64 version
LI 20 cs=0
BE 30 for i=828 to 958:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
FH 60 if cs<>14755 then print"***** data error *****":end
KP 70 rem sys 828
AF 80 end
IN 100 :
EC 1000 data 76, 74,  3,165,251,141,  2,  3,165
EP 1010 data 252,141,  3,  3, 96,173,  3,  3,201
OC 1020 data  3,240, 17,133,252,173,  2,  3,133
MN 1030 data 251,169, 99,141,  2,  3,169,  3,141
MG 1040 data  3,  3, 96,173,254,  1,133, 89,162
DM 1050 data  0,160,  0,189,  0,  2,240, 22,201
CA 1060 data 32,240, 15,133, 91,200,152, 41,  3
NG 1070 data 133, 90, 32,183,  3,198, 90, 16,249
OK 1080 data 232,208,229, 56, 32,240,255,169, 19
AN 1090 data 32,210,255,169, 18, 32,210,255,165
GH 1100 data 89, 41, 15, 24,105, 97, 32,210,255
JC 1110 data 165, 89, 74, 74, 74, 74, 24,105, 97
EP 1120 data 32,210,255,169,146, 32,210,255, 24
MH 1130 data 32,240,255,108,251,  0,165, 91, 24
BH 1140 data 101, 89,133, 89, 96
```

Listing 1b: PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI 10 rem* data loader for "verifizer 4.0" *
CF 15 rem pet version
LI 20 cs=0
HC 30 for i=634 to 754:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
OG 60 if cs<>15580 then print"***** data error *****":end
JO 70 rem sys 634
AF 80 end
IN 100 :
ON 1000 data 76,138,  2,120,173,163,  2,133,144
IB 1010 data 173,164,  2,133,145, 88, 96,120,165
CK 1020 data 145,201,  2,240, 16,141,164,  2,165
EB 1030 data 144,141,163,  2,169,165,133,144,169
HE 1040 data  2,133,145, 88, 96, 85,228,165,217
OI 1050 data 201, 13,208, 62,165,167,208, 58,173
JB 1060 data 254,  1,133,251,162,  0,134,253,189
PA 1070 data  0,  2,168,201, 32,240, 15,230,253
HE 1080 data 165,253, 41,  3,133,254, 32,236,  2
EL 1090 data 198,254, 16,249,232,152,208,229,165
LA 1100 data 251, 41, 15, 24,105,193,141,  0,128
KI 1110 data 165,251, 74, 74, 74, 74, 24,105,193
EB 1120 data 141,  1,128,108,163,  2,152, 24,101
DM 1130 data 251,133,251, 96
```

Listing 2: VERIGEN

```
PI 10 rem* data loader for "verigen" *
LN 15 rem runs on c64
LI 20 cs=0
PE 30 for i=49152 to 49410:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
KH 60 if cs<>36235 then print"***** data error *****":end
EI 70 rem sys 49152
AF 80 end
IN 100 :
EP 1000 data 169,  1,162,  8,160, 12, 32,186,255
AE 1010 data 169, 11,162,235,160,192, 32,189,255
KP 1020 data 32,192,255, 32,183,255,240,  1, 96
JB 1030 data 169,  2,162,  8,160, 13, 32,186,255
NG 1040 data 169, 13,162,246,160,192, 32,189,255
IB 1050 data 32,192,255, 32,183,255,240,  1, 96
IF 1060 data 165, 43,133,254,165, 44,133,255,160
OF 1070 data  0,162,  1, 32,198,255, 32,228,255
GJ 1080 data 32,228,255,177,254,208, 12,200,208
HB 1090 data  2,230,255,177,254,208,  8, 76,214
OA 1100 data 192,200,208,  2,230,255,200,208,  2
IG 1110 data 230,255,177,254,133,251,200,208,  2
MD 1120 data 230,255,200,208,  2,230,255,169,  0
CJ 1130 data 133,252,177,254,170,201, 32,240, 15
AI 1140 data 230,252,165,252, 41,  3,133,253, 32
KF 1150 data 228,192,198,253, 16,249,200,208,  2
AO 1160 data 230,255,138,208,226,162,  2, 32,201
MJ 1170 data 255,165,251, 41, 15, 24,105,193, 32
CL 1180 data 210,255,165,251, 74, 74, 74, 74, 24
FD 1190 data 105,193, 32,210,255,169, 32, 32,210
PL 1200 data 255,162,  1, 32,198,255, 32,228,255
AI 1210 data 72, 32,183,255,201,  0,208, 16,162
KI 1220 data  2, 32,201,255,104, 32,210,255,201
LO 1230 data 13,208,227, 76, 75,192,104, 32,204
FO 1240 data 255,169,  1, 32,195,255,169,  2, 32
LA 1250 data 195,255, 96,138, 24,101,251,133,251
HE 1260 data 96, 48, 58, 73, 78, 70, 73, 76, 44
AF 1270 data 83, 44, 82, 64, 48, 58, 79, 85, 84
LM 1280 data 70, 73, 76, 44, 83, 44, 87
```

The MANAGER Column

Don Bell
Scotland, Ontario

Letters to the Manager

This issue's application is a HOME BUDGETING PROGRAM. But first, let me respond to some letters that express common concerns.

VERSIONS: First off, make sure you have version 1.06 of THE MANAGER. Otherwise, you will be fighting some bugs in the 1.04 version. I regret that I cannot help you acquire an updated version other than suggesting you return to your dealer or bug Commodore. My regrets to David Huston of Saanichton, B.C. I hope this solves Harry Hirashima's (Ballerica MA) problem.

DISK DRIVES: THE MANAGER was only designed to run on one 1541 disk drive. Hence you cannot hook up 2 disk drives or use a 4040 disk drive. My condolences to Chris Mante of New York and Keith Adams of Mississauga, Ontario.

MORE INFORMATION ON THE MANAGER: At this time there are no plans to issue new documentation, a book or a compilation of previous articles. Unfortunately, I do not have the time to photocopy individual articles for people. Stay tuned to the Transactor though, things may change. My apologies to James Bridgewater of San Bernadino CA., David Shields of Gainesville Florida, J.F. O'Neil of Miami, Eva Gray of Galion OH, Lorne Cooke of Rose Blanche, Newfoundland.

Editor's Note: Commodore may be planning a re-release of the 64 Manager. A new manual would accompany the package but I have no official word on the subject. As Don says. . . "Stay tuned".

REPORT GENERATE & ARITHMETIC: Having problems with Report Generate or Arithmetic? In future articles I will repeat some of the in-depth instructions I gave on how to use these options but within the context of new applications.

PRINTING PROBLEMS: I'm afraid I can't help you much with printer problems. I am only familiar with the 1526 Commodore printer which I am using. Also, there is no way of speeding up/compiling the printing part of the program that I am aware of.

SEARCH FUNCTION: The Search Function does not support While loops. However, there is a way around this problem. If you want to check many different fields for the same item, simply use the OR statement to string together a number of different possibilities. You are not limited to 1 line only. In a previous article I described how a police department could use the search function on a stolen goods file to pair up goods recovered with goods stolen. Using the complex search criteria (F5) – F1 = 'OBJECT' OR F2 = 'OBJECT' OR F3 = 'OBJECT'. . .it is possible to look for the object in many different fields. This is for Jim Rendant of Chicago.

CHANGING SCREENS: How do you get from one screen to the next in Enter/Edit asks Mrs. N. A. Doninger of Huntington Beach CA. F7 for next screen (page). F8 for previous screen (page).

SCRATCHING A DATA FILE: In the Manipulate Files option there is a function called 'Scratch a Data File'. This is used to remove unwanted files from a diskette. The manual is wrong here. The prompt on your screen is 'ERASE THE MATH, DATA OR INDEX FILES' not 'ERASE THE MATH, DATA OR BOTH FILES'. Sometimes you may want to keep the data file itself with its screens, but erase the math or index files so that you can create new ones. This option allows you to do that. When entering the name of the file to scratch enter the normal filename, DO NOT include prefix characters such as DA., PT. etc. If you enter a filename that is not on the disk, the program will appear to erase the file anyway. The only verification that it has been erased is that it no longer appears on the disk directory. I had no problem getting this function to work in versions 1.04 and 1.06. If you are having trouble with this function, you can always 'kill' THE MANAGER (by powering down) and perform file deletions in BASIC. If you want to erase all the files associated with your data base file use this command:

```
open 15,8,15:print#15, " s0:??filename* " :close 15.
```

The '??.' will accept all 2 letter prefixes to the filename(rf. pt. sc. etc.) and the * will accept any characters after the filename. THE MANAGER makes all filenames 16 characters long by adding blank spaces to the end of the filename. Thus, in this case, the '*' means that any number of blank characters following the filename will be acceptable. I hope this satisfies Chester Freeman of New Rochelle, N.Y.

REPORTS FOR YOUR WORDPROCESSOR: Outputting Reports to Disk For Use in a Wordprocessor can be very useful. I had success outputting a mailing list to disk for use in the 'Paperclip' wordprocessor. The report was outputted to disk as a sequential file and then that file was loaded into the wordprocessor as a sequential file. As I do not have other wordprocessors, I cannot help you much in this regard. It's the type of thing you have to experiment with yourself. Good luck to Teri & Denis Hickethier of the Commodore Computer Users Group in Heidelberg, Germany, Michael Tesch of West Chicago.

FILE SIZE: Some users have expressed dismay at the fact that they cannot attain the maximum advertised limit of THE MANAGER, i.e. 2000 records. This is only a MAXIMUM! . The problem is not THE MANAGER, it is the limited capacity of the disk drive and diskette. If your records are reasonably small – no problem creating a file of 2000 records. However, as your records get larger they eat up more bytes on the diskette and you cannot have as many records. You may also need room on the disk for your arithmetic and report files. Report files don't have to be on the same disk, but it's easier if they are. For example, using one screen, the maximum record length I could use to create 2000 records was 78 characters. For a 100 character record length I can only create 1563 records. For a 200 character record length using 2 screens I could only create 780 records. For a 300 character record length I could only create 518 records. As a rule of thumb, to calculate the approximate number of records, divide 156,000 by the length of the record. Remember the '64 is basically a home computer, not a business machine. If you're trying to do a business application requiring a lot of records, you might need a more powerful and expensive computer system. My thanks to Frank Hancock III of Houston Texas and his church membership application for bringing this problem to my attention.

Hugh Greenwood of Vancouver, B.C. is using a MANAGER file to keep track of his slide collection. It has grown so large that he needs 2 files. He now wants to create a 3rd file which will act as an index for all of the slides in the other 2 files. He was hoping to use the Rearrange a File function in the MANIPULATE FILES option to move information from one file to another. The problem he is having is that he can only move information to his new file from one of the old files, not both of them. Also, he does not know of a way to use THE MANAGER to concatenate 2 small files. Unfortunately, I can't think of a way to solve his problem either. If anyone out there has a solution to his problem, please let me know. My only thought is a solution outside of THE MANAGER program. First, make 2 reports outputting the selected fields from the 2 slide files to disk as sequential files. Then write a BASIC program to concatenate the 2 sequential files. Then either use a wordprocessor or write a BASIC program to read the files, sort and printout whatever is required. This might be more trouble then re-entering half the data!

A Home Budgeting Application

Ever wonder where all your money goes? Ever discover too late you're in a cash flow bind that you should have seen coming? Ever want to make a serious attempt at saving some money for a change? Ever make some New Year's economic resolutions? Read on McDuff. I've got just the application for all you penny pinchers, spendaholics, budgeteers, and money maniacs! Yes, you too can be in control of your economic fate. . . or at least, predict it!

The purpose of this application is to help you plan your budget a year at a time and smooth out some of your cash problems. The real value of this application to you will be determined not so much by the design of the application itself, but by your using its structure to force yourself to think about your budget and plan ahead.

Even if you don't want to use this application, you can derive several benefits from analyzing its construction.

Many applications cry out for a combination of the benefits of a database and a spreadsheet. One of the best little functions in THE MANAGER is the ability to do 'what if' calculations just like a spreadsheet. This function is available in the Enter/Edit mode using the '=' key. This application makes extensive use of this feature to help you decide what you can afford to spend in various budget categories. This feature combined with the 'accumulate' function gives you an incredibly powerful budgeting tool. You can peer into your distant economic future, plan for it and change it when necessary.

I know some of you dread using Report Generate. You'll be glad to know that this application does not use it. How did I get around it? Reports go directly to the screen using the display positions in Arithmetic. This is not to say that you cannot go ahead and do your own reports using Report Generate.

Arithmetic! Boy have I got arithmetic for you. This application should give you some real insight into how to use the Arithmetic option to create a very powerful application.

I hope this application generates as much enthusiasm in you as it did in me. After writing my own budget programs over the last few years, I was thrilled by being able to let THE MANAGER do most of the work for me this time. Instead of wasting my time debugging code, I could instead concentrate on the real meat of the matter – the ideal design for the application itself.

Screens for Home Budgeting

Use the Create/Revise option to create 6 screens altogether. The first 3 screens are for the current month's data about budget and actual amounts. The last 3 screens are for year-to-date amounts that will be computed by the 'accumulate' function.

Don't feel any obligation to use my budget categories. Make up your own that suit your particular needs. For example, you may decide that foodout is unnecessary since it is really part of entertainment expenses. You might consider a category for bank charges, overdrafts and interest on credit card accounts. These charges are a waste of money and should be minimized.

The only thing I do suggest is that you keep the same number of categories under each section (i.e. 3 for income, 8 for fixed expenses, and 15 for variable expenses) and you keep the categories and totals on the same lines of the screens. If you don't follow these 2 rules then you'll have to modify my Arithmetic file – change the display position definitions, the output of numeric data to display positions and the counter for loops. Only more advanced users should take on this challenge.

Totals and the DIFF'CE column in all screens will be calculated by the Arithmetic file using display positions.

All budget and actual amount fields are 7 characters long and numeric, the 2 fields under 'NOTES' are 40 characters long and always alphanumeric.

Here's how to obtain screen printouts of field lengths and field numbers for your first 2 screens. Select the Enter/Edit option from the main menu, then make sure the screen is clear (if not, press Shift CLR/HOME). To display field lengths, press 'up arrow'. To display field numbers, press Shift 'up arrow'. Press 'p' to get a printout either screen. You will need these printouts of field lengths and field numbers to refer to when doing searches or designing reports.

Screen 1 is used for entering budget and actual amounts for income and fixed expense categories.

```
                    HOME BUDGET ↑       ↑

                  BUDGET   ACTUAL   DIFF'CE

INCOME
PAYCHECK          ↑   ↑    ↑   ↑    (line 6)
INTEREST          ↑   ↑    ↑   ↑
OTHER             ↑   ↑    ↑   ↑
TOTALS

FIXED EXP.
SAVINGS           ↑   ↑    ↑   ↑    (line 12)
RENT/MORT.        ↑   ↑    ↑   ↑
UTILITIES         ↑   ↑    ↑   ↑
PAYMENTS          ↑   ↑    ↑   ↑
CLUB FEES         ↑   ↑    ↑   ↑
PROP TAXES        ↑   ↑    ↑   ↑
INSURANCE         ↑   ↑    ↑   ↑
LOANS             ↑   ↑    ↑   ↑    (line 19)
TOTALS

                      NOTES
↑                                   ↑
↑                                   ↑
SCREEN 1 OF 6
```

Screen 2 is used for entering budgeted and actual variable expenses.

```
                  VARIABLE EXPENSES

                  BUDGET   ACTUAL   DIFF'CE

AUTO EXP.         ↑   ↑    ↑   ↑    (line 5)
CHAR/CHURCH       ↑   ↑    ↑   ↑
CLOTHING          ↑   ↑    ↑   ↑
DENTAL/MED'L      ↑   ↑    ↑   ↑
EDUCATION         ↑   ↑    ↑   ↑
ENTERTAIN.        ↑   ↑    ↑   ↑
FOODOUT           ↑   ↑    ↑   ↑
GAS               ↑   ↑    ↑   ↑
GROCERIES         ↑   ↑    ↑   ↑
HOUSE MAINT       ↑   ↑    ↑   ↑
PURCHASES         ↑   ↑    ↑   ↑
RECREATION        ↑   ↑    ↑   ↑
TRANSPORT'N       ↑   ↑    ↑   ↑
VACATION          ↑   ↑    ↑   ↑
OTHER             ↑   ↑    ↑   ↑
TOTAL                           (line 20)

                      NOTES
↑                                   ↑
↑                                   ↑
SCREEN 2 OF 6
```

Screen 3 gives a summary statement that is calculated in the Arithmetic file.

```
                    NET INCOME

                    BUDGET   ACTUAL   DIFF'CE

INCOME                                (line 5)

EXPENSES
  FIXED                               (line 8)
  VARIABLE

NET INCOME                            (line 11)

                      NOTES
↑                                              ↑
↑                                              ↑
SCREEN 3 OF 6
```

Screens 4 to 6 do not require any data to be entered. The note fields are there in case you wish to enter your observations about your spending habits to date. They act like report screens, simply displaying the year-to-date cumulative results of the arithmetic acting on all the month records in your file.

Screen 4 is used to display year-to-date budget and actual amounts for income and fixed expense categories. Notice that it is similar to screen 1 with the exception of the date field and totals. Using the 'accumulate' function in Enter/Edit to execute the Arithmetic file several times will produce the year-to-date numbers

```
** YEAR-TO-DATE INCOME/FIXED EXPENSES **

                    BUDGET   ACTUAL   DIFF'CE

INCOME
PAYCHECK                              (line 6)
INTEREST
OTHER


FIXED EXP.
SAVINGS                               (line 12)
RENT/MORT.
UTILITIES
PAYMENTS
CLUB FEES
PROP TAXES
INSURANCE
LOANS                                 (line 19)

                      NOTES
↑                                              ↑
↑                                              ↑
SCREEN 4 OF 6
```

Screen 5 is to display year-to-date budgeted and actual variable expenses. Again, this will be the result of using the 'accumulate' function in Enter/Edit mode.

```
** YEAR-TO-DATE VARIABLE EXPENSES **

                    BUDGET   ACTUAL   DIFF'CE

AUTO EXP.                             (line 5)
CHAR/CHURCH
CLOTHING
DENTAL/MED'L
EDUCATION
ENTERTAIN.
FOODOUT
GAS
GROCERIES
HOUSE MAINT
PURCHASES
RECREATION
TRANSPORT'N
VACATION
OTHER                                 (line 19)

                      NOTES
↑                                              ↑
↑                                              ↑
SCREEN 5 OF 6
```

Screen 6 gives a year-to-date summary income statement that is calculated in the Arithmetic file.

In addition, a statement of budgeted and actual gross savings is calculated by adding year-to-date net income and year-to-date savings. Presumably, if your net income at some point is a negative amount, then you will have to draw on your savings to cover the deficit. The amount of Gross Savings will show the effect of this. The purpose of this statement is to see how you are really doing in terms of saving money. It is also useful for predicting a cash flow bind at some time in the future. If Gross Savings is a negative amount then you're going to have to borrow money from somewhere else or run up your credit cards.

```
** YEAR-TO-DATE NET INCOME **

                    BUDGET   ACTUAL   DIFF'CE

INCOME                                (line 5)

EXPENSES
  FIXED                               (line 8)
  VARIABLE

NET INCOME                            (line 11)

    GROSS SAVINGS (NET INCOME + SAVINGS)   (line 17)

                      NOTES
↑                                              ↑
↑                                              ↑
SCREEN 6 OF 6
```

Press the back arrow key to store the screens. Remember to alter the amount fields so they are numeric.

Defining the Screen Display Positions For the Arithmetic File

Choose the Arithmetic Option from the main menu in THE MANAGER.

First we will define the display positions on all 6 screens. Screen 1 is the current month's budgeted and actual income and fixed expenses. We will now define the display positions for the DIFF'CE column and the totals line.

Use the Return key to advance within a line. Use the back arrow key to advance to the next line.

NO. OF DISPLAY POS'N ON SCREEN 1 ? 17

1. LINE? 6 COLUMN? 31 LENGTH? 9
2. LINE? 6 COLUMN? 31 LENGTH? 9
3. LINE? 8 COLUMN? 31 LENGTH? 9
4. LINE? 9 COLUMN? 11 LENGTH? 9
5. LINE? 9 COLUMN? 21 LENGTH? 9
6. LINE? 9 COLUMN? 31 LENGTH? 9
7. LINE? 12 COLUMN? 31 LENGTH? 9
8. LINE? 13 COLUMN? 31 LENGTH? 9
9. LINE? 14 COLUMN? 31 LENGTH? 9
10. LINE? 6 COLUMN? 31 LENGTH? 9
11. LINE? 6 COLUMN? 31 LENGTH? 9
12. LINE? 17 COLUMN? 17 LENGTH? 9
13. LINE? 18 COLUMN? 31 LENGTH? 9
14. LINE? 19 COLUMN? 31 LENGTH? 9
15. LINE? 20 COLUMN? 11 LENGTH? 9
16. LINE? 20 COLUMN? 21 LENGTH? 9
17. LINE? 20 COLUMN? 31 LENGTH? 9

Screen 2 is the current month's budgeted and actual variable expenses. We will now define the display positions for the DIFF'CE column and the totals line.

NO. OF DISPLAY POS'N ON SCREEN 2 ? 18

18. LINE? 5 COLUMN? 31 LENGTH? 9
19. LINE? 6 COLUMN? 31 LENGTH? 9
20. LINE? 7 COLUMN? 31 LENGTH? 9
21. LINE? 8 COLUMN? 31 LENGTH? 9
22. LINE? 9 COLUMN? 21 LENGTH? 9
23. LINE? 10 COLUMN? 31 LENGTH? 9
24. LINE? 11 COLUMN? 31 LENGTH? 9
25. LINE? 12 COLUMN? 31 LENGTH? 9
26. LINE? 13 COLUMN? 31 LENGTH? 9
27. LINE? 14 COLUMN? 31 LENGTH? 9
28. LINE? 15 COLUMN? 31 LENGTH? 9
29. LINE? 16 COLUMN? 31 LENGTH? 9
30. LINE? 17 COLUMN? 31 LENGTH? 9
31. LINE? 18 COLUMN? 31 LENGTH? 9
32. LINE? 19 COLUMN? 31 LENGTH? 9
33. LINE? 20 COLUMN? 11 LENGTH? 9
34. LINE? 20 COLUMN? 21 LENGTH? 9
35. LINE? 20 COLUMN? 31 LENGTH? 9

Screen 3 is the current month's budgeted and actual income statement. We will now define the display positions for the DIFF'CE column and the totals line.

NO. OF DISPLAY POS'N ON SCREEN 3 ? 12

36. LINE? 5 COLUMN? 11 LENGTH? 9
37. LINE? 5 COLUMN? 21 LENGTH? 9
38. LINE? 5 COLUMN? 31 LENGTH? 9
39. LINE? 8 COLUMN? 11 LENGTH? 9
40. LINE? 8 COLUMN? 21 LENGTH? 9
41. LINE? 8 COLUMN? 31 LENGTH? 9
42. LINE? 9 COLUMN? 11 LENGTH? 9
43. LINE? 9 COLUMN? 21 LENGTH? 9
44. LINE? 9 COLUMN? 31 LENGTH? 9
45. LINE? 11 COLUMN? 11 LENGTH? 9
46. LINE? 11 COLUMN? 21 LENGTH? 9
47. LINE? 11 COLUMN? 31 LENGTH? 9

Screen 4 is the year-to-date budgeted and actual income and fixed expenses. . We will now define the display positions for the BUDGET ACTUAL and DIFF'CE columns.

NO. OF DISPLAY POS'N ON SCREEN 4 ? 33

48. LINE? 6 COLUMN? 11 LENGTH? 9
49. LINE? 6 COLUMN? 21 LENGTH? 9
50. LINE? 6 COLUMN? 31 LENGTH? 9
51. LINE? 7 COLUMN? 11 LENGTH? 9
52. LINE? 7 COLUMN? 21 LENGTH? 9
53. LINE? 7 COLUMN? 31 LENGTH? 9
54. LINE? 8 COLUMN? 11 LENGTH? 9
55. LINE? 8 COLUMN? 21 LENGTH? 9
56. LINE? 8 COLUMN? 31 LENGTH? 9
57. LINE? 12 COLUMN? 11 LENGTH? 9
58. LINE? 12 COLUMN? 21 LENGTH? 9
59. LINE? 12 COLUMN? 31 LENGTH? 9
60. LINE? 13 COLUMN? 11 LENGTH? 9
61. LINE? 13 COLUMN? 21 LENGTH? 9
62. LINE? 13 COLUMN? 31 LENGTH? 9
63. LINE? 14 COLUMN? 11 LENGTH? 9
64. LINE? 14 COLUMN? 21 LENGTH? 9
65. LINE? 14 COLUMN? 31 LENGTH? 9
66. LINE? 15 COLUMN? 11 LENGTH? 9
67. LINE? 15 COLUMN? 21 LENGTH? 9
68. LINE? 15 COLUMN? 31 LENGTH? 9
69. LINE? 16 COLUMN? 11 LENGTH? 9
70. LINE? 16 COLUMN? 21 LENGTH? 9
71. LINE? 16 COLUMN? 31 LENGTH? 9
72. LINE? 17 COLUMN? 11 LENGTH? 9
73. LINE? 17 COLUMN? 21 LENGTH? 9
74. LINE? 17 COLUMN? 31 LENGTH? 9
75. LINE? 18 COLUMN? 11 LENGTH? 9
76. LINE? 18 COLUMN? 21 LENGTH? 9
77. LINE? 18 COLUMN? 31 LENGTH? 9
78. LINE? 19 COLUMN? 11 LENGTH? 9
79. LINE? 19 COLUMN? 21 LENGTH? 9
80. LINE? 19 COLUMN? 31 LENGTH? 9

Screen 5 is the year-to-date budgeted and actual variable expenses. . We will now define the display positions for the BUDGET ACTUAL and DIFF'CE columns.

NO. OF DISPLAY POS'N ON SCREEN 5 ? 45

81. LINE? 5 COLUMN? 11 LENGTH? 9
82. LINE? 5 COLUMN? 21 LENGTH? 9
83. LINE? 5 COLUMN? 31 LENGTH? 9
84. LINE? 6 COLUMN? 11 LENGTH? 9
85. LINE? 6 COLUMN? 21 LENGTH? 9
86. LINE? 6 COLUMN? 31 LENGTH? 9
87. LINE? 7 COLUMN? 11 LENGTH? 9
88. LINE? 7 COLUMN? 21 LENGTH? 9
89. LINE? 7 COLUMN? 31 LENGTH? 9
90. LINE? 8 COLUMN? 11 LENGTH? 9
91. LINE? 8 COLUMN? 21 LENGTH? 9
92. LINE? 8 COLUMN? 31 LENGTH? 9
93. LINE? 9 COLUMN? 11 LENGTH? 9
94. LINE? 9 COLUMN? 21 LENGTH? 9
95. LINE? 9 COLUMN? 31 LENGTH? 9
96. LINE? 10 COLUMN? 11 LENGTH? 9
97. LINE? 10 COLUMN? 21 LENGTH? 9
98. LINE? 10 COLUMN? 31 LENGTH? 9
99. LINE? 11 COLUMN? 11 LENGTH? 9
100. LINE? 11 COLUMN? 21 LENGTH? 9

```
101. LINE? 11   COLUMN? 31   LENGTH? 9
102. LINE? 12   COLUMN? 11   LENGTH? 9
102. LINE? 12   COLUMN? 21   LENGTH? 9
103. LINE? 12   COLUMN? 31   LENGTH? 9
104. LINE? 13   COLUMN? 11   LENGTH? 9
105. LINE? 13   COLUMN? 21   LENGTH? 9
106. LINE? 13   COLUMN? 31   LENGTH? 9
107. LINE? 14   COLUMN? 11   LENGTH? 9
108. LINE? 14   COLUMN? 21   LENGTH? 9
109. LINE? 14   COLUMN? 31   LENGTH? 9
110. LINE? 15   COLUMN? 11   LENGTH? 9
111. LINE? 15   COLUMN? 21   LENGTH? 9
112. LINE? 15   COLUMN? 31   LENGTH? 9
113. LINE? 16   COLUMN? 11   LENGTH? 9
114. LINE? 16   COLUMN? 21   LENGTH? 9
115. LINE? 16   COLUMN? 31   LENGTH? 9
116. LINE? 17   COLUMN? 11   LENGTH? 9
117. LINE? 17   COLUMN? 21   LENGTH? 9
118. LINE? 17   COLUMN? 31   LENGTH? 9
119. LINE? 18   COLUMN? 11   LENGTH? 9
120. LINE? 18   COLUMN? 21   LENGTH? 9
121. LINE? 18   COLUMN? 31   LENGTH? 9
122. LINE? 19   COLUMN? 11   LENGTH? 9
123. LINE? 19   COLUMN? 11   LENGTH? 9
124. LINE? 19   COLUMN? 21   LENGTH? 9
125. LINE? 19   COLUMN? 31   LENGTH? 9
```

Screen 6 is the year-to-date budgeted and actual income statement. We will now define the display positions for the BUDGET ACTUAL DIFF'CE columns and the TOTALS. The GROSS SAVINGS totals are displayed on line 19.

```
NO. OF DISPLAY POS'N ON SCREEN 6 ? 15
126. LINE? 6    COLUMN? 11   LENGTH? 9
127. LINE? 6    COLUMN? 21   LENGTH? 9
128. LINE? 6    COLUMN? 31   LENGTH? 9
129. LINE? 10   COLUMN? 11   LENGTH? 9
130. LINE? 10   COLUMN? 21   LENGTH? 9
131. LINE? 10   COLUMN? 31   LENGTH? 9
132. LINE? 12   COLUMN? 11   LENGTH? 9
133. LINE? 12   COLUMN? 11   LENGTH? 9
134. LINE? 12   COLUMN? 21   LENGTH? 9
135. LINE? 14   COLUMN? 31   LENGTH? 9
136. LINE? 14   COLUMN? 11   LENGTH? 9
137. LINE? 14   COLUMN? 21   LENGTH? 9
138. LINE? 19   COLUMN? 31   LENGTH? 9
139. LINE? 19   COLUMN? 11   LENGTH? 9
140. LINE? 19   COLUMN? 21   LENGTH? 9
```

Budget Calculations In the Arithmetic File

After entering all the display positions, you will be in the Edit Mode of the Arithmetic Editor.

Comments are preceded by a semicolon. I've included comments to help you understand how the arithmetic operates, you may leave them out if you wish as they are not necessary for the arithmetic to work. The arithmetic just follows the screens in order, doing the required math to get a figure for each display position.

```
* INCOME (SCREEN 1)
N3-N2 to 2D1              ;CALCULATE DIFF'CES
N5-N4 to 2D2
N7-N6 to 2D3
N2+N4+N6 to R1            ;R1 = ACT'L INC. TOTAL
R1 to 2D4
N3+N5+N7 to R2            ;R2 = BUDGET INC. TOTALS
R2 to 2D5
R2-R1 to 2D6              ;DIFF'CE
;FIXED EXPENSES
8 to R7                   ;R7 IS A FIELD # COUNTER
7 to R10                  ;R10 IS DISPLAY POS'N
WHILE R7 < 24 DO
N(R7) - N(R7+1) TO 2D(R10)
R7 + 2 TO R7
R10 + 1 TO R10
ENDWHILE
0 TO R3                   ;R3 = BUDGET FIXED EXP.
8 TO R7
WHILE R7 < 24 DO
N(R7) + R3 TO R3
R7 + 2 TO R7
ENDWHILE
R3 TO 2D15
0 TO R4                   ;R4 = ACTUAL FIXED EXP.
9 TO R7
WHILE R7 < 24 DO
N(R7) + R4 TO R4
R7 + 2 TO R7
ENDWHILE
R4 TO 2D16
R3-R4 TO 2D17             ;FIXED EXPENSE DIFF'CE
;
;VARIABLE EXPENSES (SCREEN 2)
26 TO R7
18 TO R10
WHILE R7 < 56 DO
N(R7) - N(R7+1) TO 2D(R10)
R7 + 2 TO R7
R10 + 1 TO R10
ENDWHILE
26 TO R7
0 TO R5                   ;R5 = BUDGET VAR. EXP.
WHILE R7 < 56 DO
N(R7) + R5 TO R5
R7 + 2 TO R7
ENDWHILE
R5 TO 2D33
27 TO R7
0 TO R6                   ;R6 = ACTUAL VAR. EXP.
WHILE R7 < 57 DO
N(R7) + R6 TO R6
R7 + 2 TO R7
ENDWHILE
R6 TO 2D34
R5-R6 TO 2D35             ;VAR. EXP. DIFF'CE
;
;NET INCOME FOR CURRENT MONTH RECORD (SCREEN 3)
R1 TO 2D36
R2 TO 2D37
R2-R1 TO 2D38
R3 TO 2D39
R4 TO 2D40
R3-R4 TO 2D41
R5 TO 2D42
R6 TO 2D43
R5-R6 TO 2D44
R1 - (R3+R5) TO R8
R8 TO 2D45
R2 - (R4+R6) TO R9
R9 TO 2D46
R8-R9 TO 2D47
;** YEAR-TO-DATE **
; IN THE WHILE LOOP BELOW
;ALL YEAR-TO-DATE INCOME, FIXED EXPENSES
;& VARIABLE EXPENSES ARE ACCUMULATED IN
;REGISTERS & DISPLAYED ON THE SCREEN
48 TO R10                 ;SET DISPLAY POS'N START #
2 TO R7                   ;SET START OF FIELD INDEX #
12 TO R11                 ;SET START REGISTER #
0 TO R99                  ;R99 AS TEMP. STORAGE
```

```
WHILE R10 < 126 DO
IF R7 = 24 THEN R7 + 2 TO R7    ;SKIP NOTES
ENDIF
N(R7) + R(R11) TO R(R11)
R(R11) TO 2D(R10)
N(R7 + 1) + R(R11 + 1) TO R(R11 + 1)
R(R11 + 1) TO 2D(R10 + 1)
0 TO R99                        ;R99 AS TEMP STORAGE
R(R11 + 1) - R(R(11)) TO R99
R99 TO 2D(R10 + 2)
R10 + 3 TO R10
R7 + 2 TO R7
R11 + 2 TO R11
ENDWHILE
;INCOME TOTALS YEAR-TO-DATE
R12 + R14 + R16 TO R90
R90 TO 2D126
R13 + R15 + R17 TO R91
R91 TO 2D127
R91-R90 TO 2D128
;FIXED EXPENSE TOTALS YEAR-TO-DATE
R18 + R20 + R22 + R24 + R26 + R28 + R30 + R32 TO R92
R92 TO 2D129
R19 + R21 + R13 + R25 + R27 + R29 + R31 + R33 TO R93
R93 TO 2D130
R92-R93 TO 2D131
;VARIABLE EXPENSE TOTALS YEAR-TO-DATE
;BUDGET TOTALS
34 TO R11                       ;SET START REGISTER#
1 TO R100
0 TO R94
WHILE R100 < 16 DO
R(R11) + R94 TO R94
R100 + 1 TO R100
R11 + 2 TO R11
ENDWHILE
R94 TO 2D132
;ACTUAL VARIABLE EXPENSE TOTALS
35 TO R11                       ;SET START REGISTER#
1 TO R100
0 TO R95
WHILE R100 < 16 DO
R(R11) + R95 TO R95
R100 + 1 TO R100
R11 + 2 TO R11
ENDWHILE
R95 TO 2D133
R94-R95 TO 2D134                ;DIFF'CE
;YEAR-TO-DATE NET INCOME
R90 - (R92 + R94) TO R96        ;BUDGET
R96 TO 2D135
R91 - (R93 + R95) TO R97        ;ACTUAL
R97 TO 2D136
R97-R96 TO 2D137                ;DIFF'CE
R96 + R18 TO 2D138              ;GROSS SAVINGS
R97 + R19 TO 2D139
R96 + R18-R97-R19 TO 2D140
```

Setting Up Your Budget and Entering Records

The first thing to do is look at a whole year's income and expenses at once. Go into the Enter/Edit option and enter record 1 as your year budget. Then use the "what if" feature (the = key) to plan your year budget. Enter trial amounts then press the back arrow key to let the arithmetic calculate your net income on screen 3. Remember to use F7 to advance a screen and F8 to return to the previous screen.

Keep playing with the amounts until you get the savings and net income amounts you wish to aim for. Be realistic! Overly optimistic forecasts can lead to family feuds or chronic depression. It is always better to find you saved more than you expected.

Once you're happy with the figures, screen print the first 3 screens. This year budget is your guestimate of what your year-to-date screens will show at the end of the year.

Note: We are only using the screen on record 1 to fiddle with our year budget. Record 1 will eventually be the first month of our budget.

Now on a piece of paper divide expenses by 12 to get a monthly amount. This only works of course for common monthly expenses. Large amounts such as insurance, tax payments etc. will have to entered in individual months.

Now create 12 records, 1 record for each month of your budget. Do this simply by entering the date in field 1 of each record. Put the date in this format, e.g. Jan 1, 1985 would be 850101.

Now use the first 3 screens in record 1 to play with the amounts for your month budget until you get some reasonable figures. Use the 'what if' (' = ' key) to calculate the effect of changing the amounts.

Next make global entries in the budget column of record 1 by using the Shift 'C' change function. Enter any amounts that are constant every month or you have calculated 1/12th of the yearly amount. My experience has shown that you can only change 10 fields at a time. Thus, you may have to do this operation more than once. You should now have entries in the budget column of all 12 months.

The next thing to do is to enter special amounts in the budget column that you know are due in certain months, e.g. tax or insurance payments. After doing this every month's budget column should be complete, i.e. reflect your best estimate of income and expenses for that month.

You can now use the 'accumulate' function to look at the year-to-date amounts (screens 4 to 6) for the whole year. You may decide at this time to revise some of those global entries. If you want the accumulate to only go as far as a certain month, enter the complex search string (F5) for the month you want it to stop at, e.g. up to May would be N1 < 850601

Note: You may see outrageous amounts appearing in the year-to-date display positions on your screen. This is because the accumulator keeps adding when you go from record to record. The only way to clear it and get correct amounts is to re-execute the 'accumulate' function.

One more thing - how to avoid cash flow problems. The reason for GROSS SAVINGS is to help you figure out ahead of time whether you will have enough money on hand, i.e. your paycheck and your savings account, to cover a month's expenses. Doing an 'accumulate' up to any month will tell you this, i.e. if GROSS SAVINGS is a negative amount you are in trouble. Then it's time to go back and budget more for your savings account.

You are now in a position to make entries in the ACTUAL column at the end of each month. At this point you're on your own. Improvise methods that suit your perception of what financial information is necessary in order to make this budget work for you. For example, it just occurred to me that it might be helpful if initially all ACTUAL amounts were entered ahead of time (i.e. the same as BUDGET amounts). Then each month the amounts in the ACTUAL column would be changed to reflect reality. An accumulate over the year would immediately show the effect of this month's entries on what was left of the year budget.

May your budgeting efforts in the new year add rather than subtract from your enjoyment of life!

DON'T PHONE - WRITE!

If you have questions regarding this application or you would like to submit your own "terrific" application, please write me a legible, coherent letter. If you submit an application, send it on disk or at least send screen dumps of the ENTER/EDIT screen, a hand-drawn report chart and any math and sample data. I will attempt to answer letters in this column. Write to: Don Bell, P.O Box 23, Scotland, Ontario, Canada, N0E 1R0

TransBASIC Installment #2

Nick Sullivan
Scarborough, Ont.

In the first installment of TransBASIC, Nick introduced the concept of his objective: to create a method for building custom commands and incorporate them into the BASIC command set. Also in part 1, the TransBASIC Kernel was described. To take advantage of new TransBASIC command listings, one must first obtain a copy of the TransBASIC Kernel. The Kernel is only about 500 bytes long, but the source listing of the Kernel is quite long and can't be printed each time. Volume 5, Issue 05 (Hardware & Peripherals) contains the printed listing, however The Transactor Disk for every issue will include this file, plus files from the current and all previous TransBASIC articles.

The Structure of a TransBASIC Module

The TransBASIC system is basically a long program in assembler source code with one compulsory and many optional parts. The compulsory part is called the kernel. The other parts — the optional ones — are called modules.

The kernel and all the modules are written with the PAL assembler, which uses the BASIC editor to create source code. At some point the selected modules must be merged with the kernel, and assembled. The merge routine is one that was actually written for merging BASIC. But since PAL source files are structured just like BASIC text, this merge works quite nicely (thank you again Glen Pearce).

In order for modules to be merged non-destructively with the kernel and with each other, a unique line range has been assigned to every subsection of the source code. The execution routine for each module, for instance, occupies certain lines that are reserved for it alone. Other lines have been reserved for the kernel, others for statement and function keywords, others still for equates, for routine addresses, and for link vector storage.

One of the modules given in this issue is called 'Cursor Position'. It contains two commands — the statement CURSOR, and the function CLOC. I'll use 'Cursor Position' in the following discussion to illustrate the anatomy of a TransBASIC module. This discussion will give you an idea of how to structure any modules you write on your own.

The first few lines of 'Cursor Position' are BASIC REM statements. Similar documentary lines, using as much as necessary of the line range 0 through 24, will be found in every module. The numbering and format are standardized: in your own modules you should follow the pattern as closely as you can. Incidentally, this first line range is the only one in the TransBASIC system that are not uniquely assigned. This is okay because these lines are intended for reference at the time the module containing them is merged into memory — it doesn't matter if the module after that overwrites them.

If the 'Cursor Position' module contained any equates they would come next. At the moment the only equates in use in TransBASIC are those defining the reserved zero page work area at locations two through six, and those defining operating system (Kernal) routines in the 'ADD' module. That leaves plenty of room for further equates if you want them: lines 50 through 90 should be safe.

In the next range of lines you would put any statement keywords you'll be needing in your module (function keywords come a little later on). The TransBASIC statement keyword list occupies lines 100 through 597; the CURSOR statement is entered at line 101. Note that the last character in each keyword has its high byte set. In modules you write yourself use lines 400 through 597 for your keywords.

The following 500 lines are for function keywords; the 'Cursor Position' function CLOC, for example, occurs on line 600. The organization is identical to that for statements. The safest part of this area is again the end, from 900 through 1095.

The address of the execution routine for each keyword is given on the line whose number is exactly 1000 greater than the one on which the keyword itself is entered. The numbers here are actually the routine addresses minus one, owing to the particular method by which the routines are accessed.

The only other part of a module is the block of execution routines itself. You will be entirely safe for the foreseeable future if you number your routines in the upper part of the range the BASIC line editor accepts — above 50000, say. I number the execution routines by twos in order to allow a little bit of room for revision without the necessity of using harder-to-follow multiple statement lines.

If you glance at the listing of the TransBASIC kernel (printed in last issue) you'll see at once that the lines are not all contiguous, but are scattered in various blocks throughout the range 25 to 2572. Since you won't need to make any changes to kernel lines I won't list the details of the line assignments here. In fact, you can generally ignore the kernel altogether, even though it does most of the work of linking in new commands.

TransBASIC offers probably the easiest way to extend BASIC with your own custom commands. A brief glance at some existing modules will show you just how little effort is necessary for sculpting a new one.

A typical TransBASIC command uses lots of ROM routines, particularly for expression evaluation. In the next issue we'll look at some of those routines and how to use them.

New Commands

This part of the TransBASIC column is devoted to describing the new commands that will be added each issue. The descriptions follow a standard format:

The first line gives the command keyword, the type (statement or function), and a three digit serial number.

The second line gives the line range allotted to the execution routine for the command.

The third line gives the module in which the command is included.

The fourth line (and the following lines, if necessary) demonstrate the command syntax.

The remaining lines describe the command.

CURSOR (Type: Statement Cat #: 004)
Line Range: 2574–2604
Module: CURSOR POSITION
Example: CURSOR 11
Example: CURSOR ROW,COL
Moves the cursor to specified row (0–24) and column (0–39). Column zero is assumed if no second parameter is present.

CLOC (Type: Function Cat #: 005)
Line Range: 2606–2618
Module: CURSOR POSITION
Example: IF PEEK(CLOC)<>32 GOTO 100
A quasi–variable that returns the actual memory location of the cursor.

DOKE (Type: Statement Cat #: 007)
Line Range: 2636–2672
Module: DOKE & DEEK
Example: DOKE 788,59953: REM RESET IRQ VECTOR
Pokes a 16–bit value into a pair of memory locations, the lower of which is specified in the command. The IRQ interrupt is switched out during the poke so this command may be used, as in the example, to change the vector safely.

DEEK((Type: Function Cat #: 008)
Line Range: 2674–2696
Module: DOKE & DEEK
Example: PRINT "TOP OF BASIC AT:" DEEK(55)
Returns the 16–bit value of a pair of memory locations, the lower of which is specified in the command. Standard low–high format is assumed.

SET (Type: Statement Cat #: 009)
Line Range: 2698–2706
Module: BIT TWIDDLERS
Example: SET ADDR,MASK
An 8–bit value (MASK) is ORed into the address. Bits that are set in MASK will be switched on at the specified location; other bits will be unaffected.

CLEAR (Type: Statement Cat #: 010)
Module: BIT TWIDDLERS
Example: CLEAR ADDR,MASK
Bits that are set in the 8–bit value MASK will be cleared at the specified location; other bits will be unaffected.

FLIP (Type: Statement Cat #: 011)
Line Range: 2720–2728
Module: BIT TWIDDLERS
Example: FLIP ADDR,MASK
Bits that are set in the 8–bit value MASK will be complemented at the specified location; other bits will be unaffected.

CHECK((Type: Function Cat #: 018)
Line Range: 2834–2882
Module: CHECK & AWAIT
Example: A=CHECK("AEIOU")
If there is a character in the keyboard buffer, it is tested against each character of the string argument (here "AEIOU") in turn until either a match is found or every character has been checked. If there is a match, the position of the matching character in the string (from 1 to 255) is returned. If there is no match, or if the keyboard buffer was empty, zero is returned.

AWAIT((Type: Function Cat #: 019)
Line Range: 2838–2886
Module: CHECK & AWAIT
Example: A=AWAIT("NESWQ"): ON A GOTO 10,20,30,40: END
Identical in operation to CHECK(except that it won't take no for an answer, but will wait for keyboard input and a successful match: this function never returns zero.

KEYWORDS (Type: Statement Cat #: 059)
Line Range: 4940–4980
Module: KEYWORDS
Example: KEYWORDS
All active TransBASIC keywords are printed.

```
MP  0  rem doke & deek (aug 24/84)     :
FH  1  :
JI  2  rem  1 statement,  1 function
HH  3  :
JO  4  rem keyword characters: 9
JH  5  :
NJ  6  rem keyword   routine   line    ser #
IN  7  rem s/doke    dok       2636    007
BE  8  rem f/deek(   deek      2674    008
NH  9  :
ME  10 rem u/usfp (2620/006)
PH  11 :
KD  12 rem ================================
BI  13 :
FJ  102  .asc "dokE"
GK  601  .asc "deek" : .byt $a8;deek + shifted (
BG  1102 .word dok-1
FE  1601 .word deek-1
IB  2620 usfp   ldx  #0      ;routine to convert
GM  2622        stx  $0d     ;unsigned integer
IN  2624        sta  $62     ;in .a (high byte)
OH  2626        sty  $63     ;and .y (low byte)
BB  2628        ldx  #$90    ;to floating point
FJ  2630        sec          ;in fpa #1
NH  2632        jmp  $bc49
AM  2634 ;
NO  2636 dok    jsr  $ad8a   ;get poke address
KE  2638        jsr  $b7f7   ;convert to integer
CE  2640        jsr  $aefd   ;check for comma
OC  2642        lda  $14     ;store address
GI  2644        sta  t3      ;as temp vector
DO  2646        lda  $15
KL  2648        sta  t4
FL  2650        jsr  $ad8a   ;get poke value
IF  2652        jsr  $b7f7   ;convert to integer
CM  2654        lda  $14     ;low byte to poke
JG  2656        ldy  #0      ;indirect index
LN  2658        sei          ;turn off irq
```

```
NL 2660          sta   (t3),y     ;save low byte
EI 2662          lda   $15        ;high byte to poke
LA 2664          iny              ;bump index
PM 2666          sta   (t3),y     ;save high byte
CN 2668          cli              ;irq on again
KF 2670          rts
GO 2672 ;
DN 2674 deek     jsr   $aef4      ;get val, test ')'
AP 2676          jsr   $ad8d      ;test for numeric
CA 2678          jsr   $b7f7      ;conv to integer
DI 2680          ldy   #1         ;indirect index
BM 2682          lda   ($14),y    ;get high byte
IP 2684          pha
HE 2686          dey              ;decrement index
JL 2688          lda   ($14),y    ;get low byte
BF 2690          tay
MA 2692          pla
FH 2694          jmp   usfp       ;convert to fp
OP 2696 ;
```

```
HH 0 rem bit twiddlers (sept 4/84)    :
FH 1 :
GH 2 rem 3 statements, 0 functions
HH 3 :
FE 4 rem keyword characters: 12
JH 5 :
NJ 6 rem keyword   routine  line   ser #
ID 7 rem s/set     bse      2698   009
OF 8 rem s/clear   bclr     2708   010
EN 9 rem s/flip    fli      2720   011
OH 10 :
CN 11 rem u/bprep (2730/012)
AI 12 :
LD 13 rem ================================
CI 14 :
OP 103 .asc "seTcleaRfliP"
JK 1103 .word bse-1,bclr-1,fli-1
BA 2698 bse      jsr   bprep      ;setup
MK 2700          ora   ($14),y    ;set masked bits
EC 2702          sta   ($14),y    ;store
MH 2704          rts
IA 2706 ;
GE 2708 bclr     jsr   bprep      ;setup
DP 2710          eor   #$ff       ;invert mask
NJ 2712          and   ($14),y    ;clear masked bits
AD 2714          sta   ($14),y    ;store
II 2716          rts
EB 2718 ;
KB 2720 fli      jsr   bprep      ;setup
NG 2722          eor   ($14),y    ;flip masked bits
KD 2724          sta   ($14),y    ;store
CJ 2726          rts
OB 2728 ;
GH 2730 bprep    jsr   $b7eb      ;addr to $14/15,
LP 2732          txa              ;value to .a
MI 2734          ldy   #0         ;set index to 0
MJ 2736          rts
IC 2738 ;
```

```
FO 0 rem check & await (aug 25/84)    :
FH 1 :
DH 2 rem 0 statements, 2 functions
HH 3 :
FE 4 rem keyword characters: 12
JH 5 :
NJ 6 rem keyword   routine  line   ser #
EP 7 rem f/check(  check    2834   018
NI 8 rem f/await(  await    2838   019
NH 9 :
NP 10 rem u/getter (2888/020)
PH 11 :
KD 12 rem ================================
BI 13 :
JI 602 .asc "checkawait"
IP 1602 .word check-1,await-1
OL 2834 check sec                ;set flag 'check'
IK 2836 .byte $24                ;ignore clc
```

```
FA 2838 await    clc              ;clr flag 'await'
EA 2840          php              ;save flag
FG 2842          jsr   $aef4      ;evaluate string,
NN 2844          jsr   $b6a3      ;clr descr stack
DE 2846          sta   t3         ;save str length
IB 2848          plp              ;save flag
LH 2850          ror   t4         ;minus = 'check'
BM 2852 aw1      jsr   getter     ;test key buffer
ED 2854          bit   t4         ;test flag
GD 2856          bmi   aw2        ;skip if 'check'
AF 2858          tay              ;try again if
GP 2860          beq   aw1        ;buffer empty
GA 2862 aw2      ldy   #$ff       ;initialize index
PI 2864 aw3      iny              ;bump index
MD 2866          cpy   t3         ;skip if end
AF 2868          beq   aw4        ;of string
BF 2870          cmp   ($22),y    ;test for match
CE 2872          bne   aw3
PF 2874          beq   aw5
JD 2876 aw4      bit   t4         ;test flag
CH 2878          bpl   aw1        ;loop if 'await'
EE 2880          ldy   #$ff       ;return 0
AI 2882 aw5      iny              ;pos'n in string
IP 2884          jmp   $b3a2      ;.y to fp in fac1
ML 2886 ;
OG 2888 getter   jsr   $a82c      ;test stop key
EH 2890          jmp   $e124      ;basic's getin
CM 2892 ;
```

```
LB 0 rem keywords (aug 25/84)         :
FH 1 :
AI 2 rem 1 statement, 0 functions
HH 3 :
IO 4 rem keyword characters: 8
JH 5 :
NJ 6 rem keyword   routine  line   ser #
IL 7 rem keywords  kwrds    4940   059
MH 8 :
OH 10 :
BP 116 .asc "keywordS
DF 1116 .word kwrds-1
JP 4940 kwrds    jsr   $aad7      ;print return
HP 4942          ldy   #0         ;keyword index
EK 4944 kwr1     ldx   #0         ;column counter
PP 4946 kwr2     lda   skw,y      ;get keyword char
KE 4948          beq   kwr4       ;skip at list end
PJ 4950          php              ;save status reg
DB 4952          and   #$7f       ;make lower case
FL 4954          jsr   $ab47      ;print character
OJ 4956          iny              ;bump kword index
LA 4958          inx              ;bump col counter
OH 4960          plp              ;loop if not
AI 4962          bpl   kwr2       ;end of word
LH 4964 kwr3     cpx   #$0a       ;print spaces to
GE 4966          beq   kwr1       ;pad to 10 or 20
PA 4968          cpx   #$14       ;columns
EL 4970          beq   kwr1
PL 4972          jsr   $ab3f
KE 4974          inx
NK 4976          bne   kwr3
NG 4978 kwr4     jmp   $aad7      ;print return
KO 4980 ;
```

A New Wedge For The Commodore 64

Add commands by trapping Syntax Errors

Brian Munshaw
Mississauga, Ont.

In an issue dedicated to programming aids and utilities, I thought it would be fitting to not only offer up some utilities, but to supply a superior method of adding them right to BASIC. In effect, add commands. This is the technique hinted about by Richard and Chris in previous articles, so in an effort to appease their collective curiosities, here it is.

Most of you are probably familiar with the 'chrget' wedge. It is characterized by using a special character, such as '@', '/' or '>', which precedes all commands. Both 'DOS 5.1' and 'The Manager' use this technique.

The 'chrget' wedge, however, has its drawbacks. It is what you might call a brute force method. Basically, the normal character retrieval done by BASIC is re-routed through a special routine which checks for a special 'wedging' character(s). Depending on how much checking is done, the process can slow down BASIC considerably. Some estimates indicate the slow down is around fifteen percent. Syntactic anomalies also develop. The one that comes to mind first is the problem that arises in the 'if - then' construct. In the following example the wedge command always executes.

```
if 1 = 0 then @command
```

This might seem absurd as the expression is always false, but unless this special condition is checked for, it will always execute. As a user of such a utility, to program around this, you would have to first insert a colon in front of the wedge command. Similarly, unless quote mode is checked for, the following will also invoke the wedge command.

```
print " @command "
```

The point is, unless a lot of extra checking is done to ensure against these and other problems, the average wedge takes a considerable slice of time for each and every character requested from it by BASIC. Generally, the better the wedge performs syntactically, the slower BASIC executes.

This isn't to fault the programming expertise of the original developer of the technique. He or she had few alternatives to this method on the PET. (which I assume is where the process was developed) However, the 64 is a different matter. (So is the VIC, but this article addresses itself to the 64 only! I suppose with very little effort, this technique will also work on the VIC)

The interpreter and kernal of the 64 makes extensive use of indirect jumps through vectors in RAM, more so than does the PET. The important vector in this case is the one at $0300 called 'IERROR' in the programmers reference guide.

The method I'm proposing will on the surface appear to be the same, but will exhibit none of the problems of the 'chrget' wedge. This method also requires a special preceding character for all commands, but from there on it's entirely different.

Let's consider what would happen if you just included the hypothetical command '@superdooper' to a line of BASIC. The obvious result is a syntax error when BASIC tries to execute it. But it is a very special syntax error, in that this specific syntax error occurs at the start of a statement. How can this be used to advantage? Let's examine in more detail what is actually happening. First, BASIC was happy right up to the statement before the @superdooper command. Part of the execution of that statement included positioning the text pointer (at $7a and $7b) on the first character or token of the next statement. In this case, this is the special character '@'. Since this character's ASCII value is less than 128, BASIC knows it's not a token (all tokens have the high bit set, hence must be greater than or equal to 128), and gets ready to execute an implied 'LET' statement, also known as an assignment statement. Ah, but there's the rub! It can't be an implied 'LET'! Why? Well the '@' is not a valid character for a variable name. What happens as a side effect of all this is two stack places are used, the x register is loaded with $0b signifying a syntax error, and an indirect 'jmp' is done through the 'IERROR' vector at $0300. This routine causes the appropriate error message to be printed, and execution halts.

What is of extreme importance is the consistency of these events. If you change the vector at $0300 to point to a routine that will check for your new command, and if found, will fix up the stack, scan to the next statement which allows BASIC to continue by jumping to the routine to execute the next statement, then you, my friend, have a new type of wedge, the 'error' wedge.

Most BASICs found today (excluding Commodore variants) have an 'on error goto/gosub' or 'trap' command. This is approximately the same thing. It is closer to an 'on syntax error sys' though, but this is exactly what I wanted to accomplish. A way to pass into machine code to execute the utility. It isn't a brute force method, like the 'chrget' wedge, hence doesn't slow down BASIC. It only affects processing when the command is encountered. This technique is certainly more elegant than having a separate 'sys' for each command, utility or aid and finally, it is somewhat easier than altering the tokenising and token dispatching to add commands to basic. Another nice feature comes from the fact that the kernal doesn't re-initialize the vector at $0300 with every warm start, therefore you don't have to re-engage it every time you press a run/stop - restore. This is the case when you scan for specific keyboard entries, like the function keys which some schemes do by altering the IRQ vector.

More to showcase this technique than anything else, I've included those commands which are a subset of a graphics utility I've written, which pertain to external devices, like the disk drive. These include utilities to list the directory or a program to the screen, a command to list a sequential text file to the screen, a command to alter the default output device, a command to send commands to the disk drive, two commands to read the disk status and finally a command to disable the utility.

The syntax for these commands are as follows. Please remember that the special character, '@' in this instance, must precede each command. Also as a side effect of the way the commands are parsed, each command will have a short form. The shortest acceptable will be given with the full name in each case. For those of you who don't like the preceding '@' character, and would prefer another, then poke 49152 + 10, asc(" character "). The only rule you must keep in mind when choosing a character is it must cause an automatic syntax error

and have a CBM ascii value less then 128. Possible characters do not include the mathematical operators +, –, *, /. Even though they may have an ASCII value less than 128, the editor turns them into tokens, hence will have an ASCII value greater than 127 in BASIC text.

COMMANDS :

@ (Abbreviation: none)

This command will display the disk status of the default drive.

@device, device # (Abbreviation: @de, device #)

This command will allow the user to set a default output device. When the utility is first loaded and engaged, the default device number will be set to eight. This will allow users of disk drives set to device eight to 'LOAD', 'SAVE', and 'VERIFY' without specifying a device number. If a user wishes to change this, then this command can be invoked to do so. Note, the rest of the commands will use the default device number when accessing a device, hence if you have two drives on line, remember you must alter to the appropriate device number to access the device which isn't set to default.

@dos, string (Abbreviation: @do, string)

This command will send the characters in the string (literal or variable) to the default device using secondary address fifteen. ie. send commands to the disk drive.

@dstatus, var$ (Abbreviation: @ds,var$)

This command will store the disk status in var$. Note var$ must be a string variable and works in either direct or program mode.

@kill (Abbreviation: @k)

This command disengages the utility.

@list, string (Abbreviation: @ll, string)

This command will, depending on the contents of the string literal / string variable, list either the appropriate directory or program to the screen. The device being accessed is assumed to be the default device. The syntax of the string is standard Commodore. eg:

@list, "$"

will list the directory (both if using a dual drive),

@list, "$0:*=prg"

will list the titles of all program files on drive zero and

@list, "program name"

will list the appropriate program to the screen. To cause the listing to pause, press and hold the space bar. To allow the listing to continue, release the space bar.

@print, string (Abbreviation: @?, string)

This command will print a sequential text file to the screen. The filename is held in the string, literal or variable. Use the space bar to pause.

The following is the assembler source listing and a basic program that will load and engage the utility. Once the loader has been run, you may wish to save it out with a monitor. If you do so, the code extends from $C000 to $C2C3. As you will see from the assembler listing, extensive use is made of kernal and interpreter routines. For those of you out there just starting to get your feet wet in the machine code world of the C64, this little program may help you in the set up and use of such resident routines. I hope you find the method of adding commands and the utilities useful in your own endeavours.

Listing 1: BASIC loader for the error wedge program.

```
PJ  10 rem* data loader for "error wedge" *
LI  20 cs=0
DH  30 for i=49152 to 49858:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
KG  60 if cs<>86280 then print
        "**** error in data statements****": end
DD  70 sys 49152
AF  80 end
IN  100 :
JE  1000 data  76,  52, 192, 139, 227, 165
IA  1010 data 244, 237, 245,   8,  64,   0
CA  1020 data   0,   0,   0,   0,   0,   0
MA  1030 data   0,   0,   0,   0,   0,   0
GB  1040 data   0,   0,   0,   0,   0,   0
AC  1050 data   0,   0,   0,   0,   0,   0
KC  1060 data   0,   0,   0,   0,   0,   0
ED  1070 data   0,   0,   0,   0,   0,   0
IF  1080 data   0,   0,   0,   0, 173,   0
GI  1090 data   3, 141,   3, 192, 173,   1
HJ  1100 data   3, 141,   4, 192, 169, 144
EK  1110 data 141,   0,   3, 169, 192, 141
NP  1120 data   1,   3, 173,  50,   3, 141
LG  1130 data   7, 192, 173,  51,   3, 141
CB  1140 data   8, 192, 169, 119, 141,  50
BH  1150 data   3, 169, 192, 141,  51,   3
NM  1160 data 173,  48,   3, 141,   5, 192
PN  1170 data 173,  49,   3, 141,   6, 192
CH  1180 data 169, 127, 141,  48,   3, 169
EE  1190 data 192, 141,  49,   3,  96, 174
AP  1200 data   9, 192, 134, 186, 108,   7
GC  1210 data 192, 174,   9, 192, 134, 186
DK  1220 data 108,   5, 192,  32, 115,   0
KE  1230 data  32, 158, 173,  76, 163, 182
FN  1240 data 224,  11, 240,   3, 108,   3
MB  1250 data 192,  32, 121,   0, 205,  10
HE  1260 data 192, 240,   3, 108,   3, 192
LE  1270 data 104, 104,  32, 170, 192,  76
CP  1280 data   6, 169,  32, 115,   0, 201
CA  1290 data   0, 240,   6, 201,  58, 240
IF  1300 data   2, 208,   6,  32,  45, 193
IO  1310 data  76,  87, 194, 201,  68, 208
CI  1320 data  39,  32, 115,   0,  72,  32
MK  1330 data  45, 193, 104, 201,  69, 208
OD  1340 data   7,  32, 241, 183, 142,   9
GM  1350 data 192,  96, 201,  79, 208,   3
LA  1360 data  76,  99, 193, 201,  83, 240
CL  1370 data   3, 108,   3, 192,  32,  45
IA  1380 data 193,  76, 116, 194, 201,  75
IP  1390 data 208,  40,  32,  45, 193, 173
FJ  1400 data   3, 192, 141,   0,   3, 173
DK  1410 data   4, 192, 141,   1,   3, 173
AJ  1420 data   7, 192, 141,  50,   3, 173
PJ  1430 data   8, 192, 141,  51,   3, 173
NK  1440 data   5, 192, 141,  48,   3, 173
NP  1450 data   6, 192, 141,  49,   3,  96
FO  1460 data 201, 155, 208,   6,  32,  45
EB  1470 data 193,  76, 132, 193, 201, 153
NP  1480 data 208,   6,  32,  45, 193,  76
FC  1490 data  50, 194, 162,  11, 108,   3
KN  1500 data 192,  32, 121,   0, 201,   0
MG  1510 data 240,  14, 201,  44, 240,  10
OD  1520 data 201,  58, 240,   6,  32, 115
EH  1530 data   0,  76,  48, 193,  96,  32
DF  1540 data 204, 255, 169, 127,  76, 195
```

```
MI 1550 data 255,  32, 135, 192,  32, 189
KI 1560 data 255, 164, 253, 169, 127, 174
MF 1570 data   9, 192,  32, 186, 255,  32
PJ 1580 data 192, 255, 162, 127,  76, 198
FP 1590 data 255, 173,   9, 192,  32, 177
DI 1600 data 255, 169, 111,  32, 147, 255
OI 1610 data  32, 135, 192, 134, 253, 132
DK 1620 data 254, 170, 160,   0, 177, 253
AK 1630 data  32, 168, 255, 200, 202, 208
GI 1640 data 247,  76, 174, 255, 169,   0
EC 1650 data 133, 253,  32,  75, 193,  32
CN 1660 data 207, 255,  32, 207, 255, 169
LI 1670 data   0, 141,  11, 192, 169,  13
HN 1680 data  32, 210, 255,  32, 207, 255
KP 1690 data  32, 207, 255,  32, 207, 255
MN 1700 data 133, 253,  32, 207, 255, 133
OI 1710 data 254,  32, 183, 255, 240,   3
BC 1720 data  76,  67, 193, 165, 254, 164
ED 1730 data 253,  32, 145, 179,  32, 221
HB 1740 data 189, 160,   1, 185,   0,   1
HF 1750 data 240,   6,  32, 210, 255, 200
IB 1760 data 208, 245, 169,  32,  32, 210
NF 1770 data 255,  32, 225, 255, 240,  52
MA 1780 data 201, 239, 240, 247,  32, 207
HM 1790 data 255, 240, 180,  48,  20,  32
EC 1800 data 210, 255, 201,  34, 208, 233
LG 1810 data 238,  11, 192, 173,  11, 192
LC 1820 data  41,   1, 141,  11, 192,  16
JO 1830 data 220, 174,  11, 192, 240,   6
MI 1840 data  32, 210, 255,  76, 207, 193
KH 1850 data 201, 255, 240, 246,  32,  11
LF 1860 data 194,  76, 207, 193,  76,  67
JP 1870 data 193, 160,   0,  56, 233, 128
NJ 1880 data 240,  15, 170, 169, 255, 200
KE 1890 data 185, 158, 160,  48,   2, 208
HK 1900 data 248, 202, 208, 245, 200, 185
ON 1910 data 158, 160,  48,   6,  32, 210
AK 1920 data 255, 200, 208, 245,  56, 233
CI 1930 data 128,  76, 210, 255, 169,   2
GE 1940 data 133, 253,  32,  75, 193,  32
NN 1950 data 207, 255, 170,  32, 183, 255
MA 1960 data 240,   3,  76,  67, 193, 138
HP 1970 data  32, 210, 255,  32, 225, 255
IA 1980 data 208,   3,  76,  67, 193, 201
DB 1990 data 239, 208, 228, 240, 242, 173
MG 2000 data   9, 192,  32, 180, 255, 169
BL 2010 data 111,  32, 150, 255, 169,   0
DF 2020 data 133, 144,  32, 165, 255,  32
PB 2030 data 210, 255,  32, 183, 255, 240
FA 2040 data 245,  76, 171, 255, 173,   9
MF 2050 data 192,  32, 180, 255, 169, 111
EO 2060 data  32, 150, 255, 169,   0, 133
HG 2070 data 144, 168,  32, 165, 255, 153
PF 2080 data  12, 192, 200,  32, 183, 255
LK 2090 data 208,   4, 192,  40, 208, 240
JL 2100 data  32, 171, 255, 136, 152,  72
PF 2110 data  72,  32, 115,   0,  32, 139
CB 2120 data 176, 133,  73, 132,  74,  32
OH 2130 data 163, 182, 104,  32, 117, 180
EH 2140 data 160,   2, 185,  97,   0, 145
NK 2150 data  73, 136,  16, 248, 200, 104
NA 2160 data 170, 185,  12, 192, 145,  98
MJ 2170 data 200, 202, 208, 247,  96
```

Error Wedge Source Listing

```
OC 100  sys 700                         ;assembled on pal 64
GN 110  .opt oo
OO 120  ;
LO 130          *       = $c000         ;program origin
CA 140  ;
MA 150  ;
OD 160  ;constants
AC 170  ;
AH 180  list    = $9b                   ;list token val
MF 190  print   = $99                   ;print token val
FK 200  illquan = $0e                   ;error values
JL 210  syntax  = $0b                   ;  "
EG 220  status  = $90                   ;status byte
AF 230  comma   = ","                   ;ascii of comma
DE 240  colon   = ":"                   ;ascii of colon
PB 250  eol     = 0                     ;end of line
HF 260  stack   = $0100                 ;bottom 65xx stk
GD 270  keytable = $a09e                ;keyword table
OI 280  ;
KM 290  ; interpreter routines used
CK 300  ;
DK 310  chrget  = $0073                 ;get next char
GE 320  chrgot  = $0079                 ;get last char
EL 330  ready   = $a474                 ;to 'ready
CB 340  scannext = $a906                ;to next statement
AK 350  fltexp  = $ad9e                 ;eval gen exp
BD 360  lookup  = $b08b                 ;variable
HB 370  intflt  = $b391                 ;int->flt conv.
FK 380  strres  = $b475                 ;res spc 4 string
BP 390  discrd  = $b6a3                 ;unwanted string
AB 400  exp     = $b7f1                 ;int exp in x-reg
FK 410  fltasc  = $bddd                 ;flt->ascii conv
KB 420  ;
DB 430  ; kernal routines used
OC 440  ;
HD 450  close   = $ffc3                 ;opened file
JN 460  chkin   = $ffc6                 ;set input
HH 470  chrout  = $ffd2                 ;output to file
NI 480  setlfs  = $ffba                 ;set logical 1,2
OG 490  open    = $ffc0                 ;open file
BJ 500  setnam  = $ffbd                 ;set file name
HC 510  readst  = $ffb7                 ;get status
BB 520  chrin   = $ffcf                 ;get from device
PL 530  clrchn  = $ffcc                 ;clear i/o
DG 540  stop    = $ffe1                 ;scan stop
LH 550  talk    = $ffb4                 ;send talk
LC 560  tksa    = $ff96                 ;send sa for talk
KK 570  untalk  = $ffab                 ;send untalk
LD 580  acptr   = $ffa5                 ;get serial byte
NM 590  listen  = $ffb1                 ;send listen
MF 600  lsnsa   = $ff93                 ;send sa - listen
EK 610  unlsn   = $ffae                 ;send unlisten
GA 620  ciout   = $ffa8                 ;send serial byte
MO 630  ;
JI 640  ; ram indirect vectors altered
AA 650  ;
EN 660  ierror  = $0300                 ;errors
FE 670  iload   = $0330                 ;loads
HF 680  isave   = $0332                 ;saves
IC 690  ;
BC 700  ; zero page used
MD 710  ;
LM 720  varaddr = $49                   ;address of var
JK 730  flacc1  = $61                   ;flt. pt. acc. #1
OE 740  curlin  = $39                   ;current line
PN 750  curdev  = $ba                   ;current device
IN 760  temp    = $fd                   ;temp storage
IH 770  ;
LP 780          jmp  start              ;around storage
MI 790  ;
CO 800  ; general memory storage
AK 810  ;
FE 820  errvector  .word 0              ;old err vector
KJ 830  loadvector .word 0              ;old load  "
HK 840  savevector .word 0              ;old save  "
CJ 850  curdevnum .byte 8               ;default device#
MJ 860  mychar     .asc "@"             ;err wedge char
NM 870  quoteflag  .byte 0              ;boolean
GH 880  tempstr    * = * + 40           ;space for string
AP 890  ;
KO 900  start   = *
EA 910  ;
IK 920          lda  ierror             ;alter vectors
MM 930          sta  errvector          ;in ram to new
FB 940          lda  ierror + 1         ;routines.
CG 950          sta  errvector + 1      ;save the old
CC 960          lda  #< onerror         ;values.
OF 970          sta  ierror             ;re-direct to
ID 980          lda  #> onerror         ;new routines
MC 990          sta  ierror + 1
BM 1000         lda  isave
LP 1010         sta  savevector
HJ 1020         lda  isave + 1
BN 1030         sta  savevector + 1
KI 1040         lda  #< setsave
BD 1050         sta  isave
KJ 1060         lda  #> setsave
HA 1070         sta  isave + 1
PA 1080         lda  iload
KE 1090         sta  loadvector
FO 1100         lda  iload + 1
AC 1110         sta  loadvector + 1
IN 1120         lda  #< setload
PH 1130         sta  iload
IO 1140         lda  #> setload
FF 1150         sta  iload + 1
EH 1160         rts
IA 1170 ;
NF 1180 setsave = *
MB 1190 ;
DO 1200         ldx  curdevnum
AK 1210         stx  curdev
KA 1220         jmp  (savevector)
EE 1230 ;
IJ 1240 setload = *
IF 1250 ;
PB 1260         ldx  curdevnum
MN 1270         stx  curdev
EE 1280         jmp  (loadvector)
AI 1290 ;
DJ 1300 getstring = *
CM 1310 ;   string input and discard
KE 1320         jsr  chrget
FN 1330         jsr  fltexp
MK 1340         jmp  discrd
```

```
1350 ;
1360 onerror     = *
1370 ;   this is it folks
1380             cpx  #syntax
1390             beq  rightype
1400             jmp  (errvector)
1410 ;
1420 rightype    = *
1430 ;
1440             jsr  chrgot
1450             cmp  mychar
1460             beq  myerror
1470             jmp  (errvector)
1480 ;
1490 myerror     = *
1500 ;
1510             pla                  ;pop unnecessary
1520             pla                  ;stuff
1530             jsr  parse
1540             jmp  scannext        ;give back to basic
1550 ;
1560 ; short and sweet ain't it
1570 ;
1580 parse       = *
1590 ;   command parser
1600             jsr  chrget          ;a straight character
1610             cmp  #eol            ;by character search
1620             beq  show            ;through basic text
1630             cmp  #colon          ;to identify command.
1640             beq  show            ;not elegant, but
1650             bne  next1           ;functional.
1660 ;
1670 show        = *
1680 ;
1690             jsr  scan
1700             jmp  showds
1710 ;
1720 next1       = *
1730 ;
1740             cmp  #"d"            ;command start with a
1750             bne  next2           ;"d" no => try another
1760 ;
1770             jsr  chrget          ;command starts with
1780             pha                  ;"d"
1790             jsr  scan
1800             pla
1810             cmp  #"e"            ;check second char.
1820             bne  other1
1830 ;
1840             jsr  exp             ;set new default
1850             stx  curdevnum       ;device number.
1860             rts
1870 ;
1880 other1      = *
1890 ;
1900             cmp  #"o"            ;try another second
1910             bne  other2          ;character
1920 ;
1930             jmp  doscomm         ;send dos command
1940 ;
1950 other2      = *
1960 ;
1970             cmp  #"s"            ;last chance 4 command
1980             beq  dsfunc          ;starting with a "d"
1990             jmp  (errvector)     ;invalid command
2000 ;
2010 dsfunc      = *
2020 ;
2030             jsr  scan
2040             jmp  inputds         ;input disk status
2050 ;
2060 next2       = *
2070 ;
2080             cmp  #"k"
2090             bne  next3
2100 ;
2110             jsr  scan            ;kill utility
2120             lda  errvector       ;i.e. restore
2130             sta  ierror          ;original vectors
2140             lda  errvector + 1
2150             sta  ierror + 1
2160             lda  savevector
2170             sta  isave
2180             lda  savevector + 1
2190             sta  isave + 1
2200             lda  loadvector
2210             sta  iload
2220             lda  loadvector + 1
2230             sta  iload + 1
2240             rts
2250 ;
2260 next3       = *
2270 ;
2280             cmp  #list
2290             bne  next4
2300 ;
2310             jsr  scan            ;list program or
2320             jmp  dolist          ;directory routine
2330 ;
2340 next4       = *
2350 ;
2360 ; print text file
2370             cmp  #print          ;routine
2380             bne  thatsall
2390 ;
2400             jsr  scan
2410             jmp  doprint
2420 ;
2430 thatsall    = *
2440 ;
2450             ldx  #syntax         ;that's all for now
2460             jmp  (errvector)     ;real syntax error
2470 ;
2480 scan        = *
2490 ;
2500             jsr  chrgot          ;get last character
2510 ;
2520 scanmore    = *
2530 ;
2540             cmp  #eol            ;strip out extra
2550             beq  out             ;text. exit on eol
2560             cmp  #comma          ;comma or colon
2570             beq  out
2580             cmp  #colon
2590             beq  out
2600             jsr  chrget
2610             jmp  scanmore
2620 ;
2630 out         = *
2640 ;
2650             rts
2660 ;
2670 closeinput  = *
2680 ;
2690             jsr  clrchn          ;close an input
2700             lda  #127            ;file.
2710             jmp  close
2720 ;
2730 openinput   = *
2740 ;
2750             jsr  getstring       ;open input file
2760             jsr  setnam
2770             ldy  temp            ;get secondary
2780             lda  #127            ;file # 127 used!
2790             ldx  curdevnum
2800             jsr  setlfs
2810             jsr  open
2820             ldx  #127
2830             jmp  chkin
2840 ;
2850 doscomm     = *
2860 ;
2870             lda  curdevnum       ;get device number
2880             jsr  listen          ;tell it to listen
2890             lda  #$6f            ;$6f = ($0f or $60)
2900             jsr  lsnsa           ;listen secondary
2910             jsr  getstring       ;get command string
2920             stx temp : sty temp + 1
2930             tax : ldy #0
2940 ;
2950 disklp      = *                  ;send command
2960 ;
2970             lda (temp),y : jsr ciout
2980             iny : dex : bne disklp
2990             jmp  unlsn
3000 ;
3010 ;
3020 dolist      = *
3030 ;
3040             lda  #0              ;give secondary of
3050             sta  temp            ;zero
3060             jsr  openinput
3070             jsr  chrin           ;waste start address
3080             jsr  chrin
3090 ;
3100 dirloop     = *
3110 ;
3120             lda  #0              ;initialize quote
3130             sta  quoteflag       ;mode
3140             lda  #13             ;send a return to the
3150             jsr  chrout          ;screen
3160             jsr  chrin           ;waste pointers
3170             jsr  chrin           ;"
3180             jsr  chrin           ;get #< block count
3190             sta  temp            ;/ linenum
3200             jsr  chrin           ;get #> block count
3210             sta  temp + 1
3220             jsr  readst          ;end of file"?
3230             beq  * + 5           ;no => get more
3240             jmp  closeinput      ;yes => exit
3250             lda  temp + 1        ;convert integer prg.
3260             ldy  temp            ;size to floating
3270             jsr  intflt          ;point
3280             jsr  fltasc          ;convert float-ascii
3290             ldy  #1
3300 ;
3310 proutnum    = *
3320 ;
3330             lda  stack,y         ;print out ascii
3340             beq  addspace        ;string from stack
3350             jsr  chrout
3360             iny
3370             bne  proutnum
3380 ;
3390 addspace    = *
3400 ;
3410             lda  #" "            ;pad out to improve
3420             jsr  chrout          ;appearance
3430 ;
3440 dirline     = *
3450 ;
3460             jsr  stop            ;check for stop key
3470             beq  dirend          ;yes => quit
3480             cmp  #$ef            ;check for spacebar
3490             beq  dirline         ;yes => pause list
3500             jsr  chrin           ;get rest of line
3510             beq  dirloop
3520             bmi  iskey           ;bit7 set => token "?
3530             jsr  chrout          ;not token, print it
3540             cmp  #34             ;was it a quote print
3550             bne  dirline         ;no => continue
3560             inc  quoteflag       ;yes => quote mode
3570             lda  quoteflag       ;i.e. toggle
3580             and  #1              ;boolean quoteflag
3590             sta  quoteflag
3600             bpl  dirline         ;branch always
3610 ;
3620 iskey       = *
3630 ;
3640             ldx  quoteflag       ;quotemode "?
3650             beq  dokey           ;yes => print token
3660 ;
3670 dontdok     = *
3680 ;
3690             jsr  chrout          ;no => shifted char
3700             jmp  dirline         ;return to loop
3710 ;
3720 dokey       = *
3730 ;
3740             cmp  #255            ;is it pi
3750             beq  dontdok         ;not in token list
3760             jsr  listkey
3770             jmp  dirline         ;return to loop
3780 ;
3790 dirend      = *
3800 ;                                ;close file
3810             jmp  closeinput
3820 ;
3830 listkey     = *
3840 ;
3850             ldy  #0              ;token in acc
3860             sec
3870             sbc  #$80            ;normalize token range
3880             beq  prkey           ;to 0 - 127
3890             tax
3900             lda  #$ff
3910 ;
3920 sckeytab    = *
3930 ;
3940             iny                  ;get to the right
3950             lda  keytable,y      ;token in the
3960             bmi  * + 4           ;table
3970             bne  sckeytab
3980             dex
3990             bne  sckeytab
4000             iny
4010 ;
4020 prkey       = *
4030 ;
4040             lda  keytable,y      ;found it now
4050             bmi  endprkey        ;print it out
4060             jsr  chrout          ;last character
4070             iny                  ;of token in table
4080             bne  prkey           ;has bit7 set
4090 ;
4100 endprkey    = *
4110 ;
4120             sec                  ;unshift last char
4130             sbc  #$80            ;print and return
4140             jmp  chrout          ;after chrout
4150 ;
4160 doprint     = *
4170 ;
4180             lda  #2              ;use secondary
4190             sta  temp            ;of two
4200             jsr  openinput       ;open the file
4210 ;
4220 listloop    = *
4230 ;
4240             jsr  chrin           ;get character
4250             tax                  ;save in x reg.
4260             jsr  readst          ;check ds
4270             beq  * + 5           ;not done
4280             jmp  closeinput      ;done
4290             txa                  ;everthing ok
4300             jsr  chrout          ;then print it
4310 ;
4320 listdelay   = *
4330 ;   should we pause "?"
4340             jsr  stop            ;check for stop
4350             bne  * + 5           ;nope => more
4360             jmp  closeinput      ;stop !
4370             cmp  #$ef            ;is it spacebar
4380             bne  listloop        ;no => go back
4390             beq  listdelay       ;yes => more delay
4400 ;
4410 showds      = *
4420 ;  show disk status to screen
4430             lda  curdevnum       ;get drive number
4440             jsr  talk            ;talk to me drive
4450             lda  #$6f            ;$6f=($0f or $60)
4460             jsr  tksa            ;send secondary
4470             lda #0 : sta status
4480 ;
4490 dsloop      = *
4500 ;
4510             jsr  acptr           ;get serial
4520             jsr  chrout          ;print it
4530             jsr  readst          ;check st
4540             beq  dsloop          ;ok => do more
4550             jmp  untalk          ;shut up drive
4560 ;
4570 inputds     = *                  ;read disk status
4580 ;                                ;& assign to var
4590             lda  curdevnum       ;get device number
4600             jsr  talk            ;talk to me drive
4610             lda  #$6f            ;$6f = ($0f or $60)
4620             jsr  tksa            ;send secondary
4630             lda #0 : sta status  ;force zero
4640             tay
4650 ;
4660 rdsloop     = *
4670 ;
4680             jsr  acptr           ;get serial
4690             sta tempstr,y : iny  ;save tempstr
4700             jsr  readst          ;check st
4710             bne  outrds          ;no => exit
4720             cpy  #40             ;limit length
4730             bne  rdsloop         ;to 40 chars.
4740 ;
4750 outrds      = *
4760 ;
4770             jsr  untalk          ;shut up drive
4780 ;
4790 ; store it out
4800 ;
4810             dey:tya:pha:pha      ;save len twice
4820             jsr  chrget          ;advance textptr
4830             jsr  lookup          ;lookup var
4840             sta  varaddr         ;store address
4850             sty  varaddr + 1
4860             jsr  discrd          ;get rid of it
4870             pla                  ;remember length
4880             jsr  strres          ;reserve space for str
4890             ldy  #2
4900 ;
4910 store       = *
4920 ;
4930             lda  flacc1,y        ;3 bytes are
4940             sta  (varaddr),y     ;length & address
4950             dey
4960             bpl  store
4970             iny                  ;set y to 0
4980             pla : tax            ;remember length again
4990 ;
5000 transfer    = *                  ;from temp to var
5010 ;
5020             lda  tempstr,y
5030             sta  (flacc1 + 1),y
5040             iny : dex
5050             bne  transfer
5060             rts
5070 ;
5080 .end
5090 end
```

The Commodore 64 Keyboard Part 2:

Aubrey Stanley
Mississauga, Ont.

KEYWIZARD – An Amazing Keyboard Driver

Key Wizard is an alternative to the Kernel Keyboard Driver which I described in Part 1 (last issue). But it is more than just a driver. Many other features have been added. It will literally bring joy to your fingertips!

Part 2 is for everybody. It describes the functions and gives the BASIC Loader. It is a long program, so typing it in may be a chore. However, with the new listing format in The Transactor you should be safe from errors.

In Part 3 I will be presenting the source. I intend to build more functions into this program to suit my own particular needs. I'm sure many of you will wish to do the same. If you have any good ideas and/or source for some really useful routines, please submit them to the TRANSACTOR.

The Key Wizard system gives you true N–KEY ROLLOVER capability. This means that keys are displayed as they are pressed even if previous keys have not as yet been released. The 'N' factor can be set by the user to any value. But remember that the Commodore keyboard matrix often generates false values when three or more keys are pressed together. For this reason it may be desirable to leave this count at 2, its default setting. Even a 2–KEY ROLLOVER vastly improves the productivity of keyboard input and allows you to repeat the two keys pressed; for example, when you wish to scroll the cursor diagonally across the screen using Cursor Down and Cursor Right

Key Wizard is totally transparent to software that uses the standard driver for keyboard input. The interface is identical to that of the standard driver. I have used it with development packages like 'POWER', 'MOREPOWER' and 'PAL' to further enhance the speed and ease of program development.

In a standard memory configuration using BASIC, Key Wizard needs only 24 bytes of memory which may be located at any suitable address. Normally, Key Wizard uses the address range from 53224 to 53247 ($CFE8 to $CFFF). The rest of the program is tucked away in the normally unused block of RAM behind the BASIC ROM. Therefore, adding functions to this program will not take away from the useful memory space.

Key Wizard is an invaluable keyboard driver for program development as it greatly enhances the editing capabilities of the standard screen editor. Multiple features have been built into the program and the potential exists to incorporate many more. For complete flexibility, the user is allowed to map his own routine to any key.

For those who would like to rearrange the keyboard, Key Wizard allows each and every key to be re–configured (except for the Control, Shift and Commodore Keys). Change one or two keys, or the entire keyboard for that matter. *(Can you say, "Dvorak"? KJH.)*

Assigning keys to strings is another capability built into the Key Wizard system. Map your most used BASIC keywords, commands, etc, to keys of your choice and have them displayed when the keys are pressed.

Changing the colour of border, background or text can be done by hitting a key with Key Wizard. Cycle through the combinations until you find one that suits you best.

Key Wizard allows you to save your keyboard profile on diskette after you have mapped the keyboard to your own particular requirements. More than one profile may be saved, to be loaded in at any future time when needed.

The BASIC Loader

the Key Wizard system is provided in the form of a BASIC Loader program called "KEYWIZ–64". This program does the following:

1. Puts the Key Wizard machine–language into memory.

This is the only step that is performed when the program is simply RUN. As listed, the machine–language program is POKEd into memory from the DATA statements in two separate blocks:

(a) LINES 200 and 201 contain the IRQ Vector Entry Routine, 24 bytes. Normally this is loaded into location 53224. But since it is totally relocatable, you may wish to move it to some other location. To do so, simply assign the new address to the variable 'I' in LINE 52. Also in LINE 52 is the value of the N–KEY ROLLOVER count, NK. This is assigned a value of 2. You may change it to any value, but remember that the Commodore Keyboard hardware can give false values when more than two keys are pressed simultaneously. For this reason, I would leave the value at 2.

A word of warning for 'POWER' users. I found that relocating the program in the Cassette Buffer interferes with POWER'S String Search facility.

(b) LINE 300 on contains the Key Wizard machine–language program, 2.5K bytes. This is loaded behind the BASIC ROM, starting at 41216 ($A100).

2. Saves Key Wizard on diskette.

The routine at LINE 70 will save the Key Wizard machine–language program to disk as a program file (this is done after execution of Key Wizard's save function as explained later). Saving the program as a file reduces the loading time the next time KEYWIZ–64 is run, since the machine–language program need not be POKEd into memory from DATA statements.

3. Saves your Keyboard Profile on diskette.

After you have rearranged the keyboard, defined strings, etc, you may save the keyboard profile permanently on diskette using the routine at LINE 80.

Loading The Program And Keyboard Profile

First, use KEYWIZ–64 to save the Key Wizard machine–language program on diskette. If you have not not yet done this, refer to the section on Saving The Program. This will considerably reduce your loading time in the future.

LINE 50 tells KEYWIZ-64 how to go about loading the Key Wizard system. There are two variables associated with this process. 'TB$' is assigned the KEYBOARD PROFILE name, and 'PG$' the PROGRAM Name. Initially TB$ = "KERNEL" and PG$ = "KPRG". These default values tell KEYWIZ-64 that (a) the standard keyboard arrangement as provided by the KERNEL is to be used (as opposed to a profile from disk) and (b) to load Key Wizard from the DATA statements in lines 300 onwards rather than from a disk file. This is the set-up used the first time KEYWIZ-64 is run; after that, the Key Wizard machine-language program is loaded from a disk file.

If you assign any OTHER name to TB$ in LINE 50, then KEYWIZ-64 will load the Keyboard Profile from disk using TB$ as the filename.

If you assign any OTHER name to PG$ in LINE 50, then "KEYWIZ-64" will load the Key Wizard machine language program from disk using PG$ as the filename.

Once you have saved the program on diskette, you should modify the value of PG$ in LINE 50 to the saved name. Then you may delete Lines 300 onwards as they will no longer be required. You'll need the routines at the beginning of KEYWIZ-64 to make new keyboard profiles - eliminating all those unnecesary DATA statements will considerably reduce the loading time. You should make a Backup Version of KEYWIZ-64 before you start to modify it.

If you do not have a utility to delete BASIC lines automatically, I would suggest that you save KEYWIZ-64 after you have entered all lines up to Line 300. After entering the remaining DATA statements, save "KEYWIZ-64 W/DATA" as a BACKUP version on another diskette. Finally, follow the instructions (below) to save the machine language for Key Wizard as a program file on the diskette that contains your original, shortened version. From then on use your short version.

Saving The Program And Keyboard Profile

After the loader is RUN once, the Key Wizard machine-language program can be saved as a PRG file on disk.

The name of the PROGRAM file under which the Key Wizard machine-language program is saved is assigned to the variable PG$ in LINE 72. I have used the name "KWIZ". You may change this to any other name if you like. Then type:

CONTROL and SPACEBAR

Each time you type this command, Key Wizard will copy either the Program or the Keyboard Profile into the memory area beginning at 49152 ($C000). So be sure that you do not have anything running in this area when you enter this command. Key Wizard will display an appropriate message to tell you which operation was performed. So if you see the message, "SAVE TB", you should then enter the command again and "SAVE PG" will be displayed the next time. Key Wizard alternates between the two each time the command is entered. Type:

RUN 70

and the Key Wizard machine-language program will be saved on diskette.

The name of the KEYBOARD PROFILE under which the tables are saved is assigned to the variable TB$ in LINE 82. I have used the name "KWIZ.TB". You may change this to any other name if you like. Then enter the above command until you see the message, "SAVE TB". Type:

RUN 80

and the current keyboard profile will be saved on diskette.

Now save the shortened version of KEYWIZ-64 (LINES up to 250) with the new assignments for PG$ and TB$. Each time this program is RUN, the Key Wizard program will be loaded from disk directly into memory.

You may wish to save a variety of Keyboard Profiles to suit different applications. Remember to use different names in this case and to assign the desired name to TB$ in LINE 50 before running KEYWIZ-64.

To recap the use of the BASIC KEYWIZ-64 program: the first time you run the program, it should be used in exactly the same form as it appears in the listing. When RUN, it will place the Key Wizard machine-language program and default keyboard profile in memory. After that, a new version of "KEYWIZ-64" can be created, one without the DATA statements (lines 300 to end), and with the assignments in line 50 changed to:

tb$ = "kwiz.tb" : pg$ = "kwiz"
(assuming the default filenames)

This new version will be used in the future to load and enable The Key Wizard system, but first the Key Wizard machine-language program and keyboard profile must be saved to disk.

The Key Wizard machine-language program is saved to disk with:

CTRL/SPACE, then RUN 70

and the current keyboard profile is saved with:

CTRL/SPACE, then RUN 80.

(The CTRL/SPACE combination acts as a toggle and alternately commits the program or the profile to memory where it can be read by the save routines)

Changing The Screen Colours

The Screen Colours may be changed to any of the 16 standard colours by using the following commands:

CONTROL and HOME	for Border
CONTROL and DELETE	for Screen
CONTROL and RETURN	for Text

Use a particular command repeatedly until the desired colour is displayed. THESE COMMANDS ONLY WORK IN DIRECT MODE.

Repeat Mode

In repeat mode, all keys are repeated in both Direct and RUN mode. To toggle repeat mode, use:

CONTROL and SHIFT

Normally repeating keys (Cursor, Spacebar, etc.) are not affected.

Alternative INSERT/DELETE Mode

The alternative Insert puts the BASIC editor into a permanent "insert mode", where space is automatically opened up in text as characters are entered. In other words, all characters to the right of the cursor move forward as you type, until the BASIC line is full (80 characters long).

The alternative Delete function uses the DELETE key to delete the character under the cursor. All characters to the right of the deleted

position, up to the end of the BASIC Line, are automatically moved left by one position. To delete to the end of the BASIC Line, just hold down the DELETE key until all the characters are erased.

This mode is toggled on and off by successive entries of the keys:

CONTROL and COMMODORE

THIS COMMAND MAY BE ENTERED AT ANY TIME, BUT THE ALTERNATIVE INSERT/DELETE MODE WILL ONLY BE IN EFFECT DURING DIRECT MODE.

Strings

You may assign a string to a key or "undo" a previously defined string by entering the command:

CONTROL and CURSOR DOWN

Key Wizard will display the message:

STRING ?

In case you have entered the command in error, type RETURN to abort. Otherwise enter the key you want to make into a string or to undo from a previous assignment. All values are permissible – normal, shifted, Commodore or control. There are some restrictions, for example you cannot assign a key that has already been assigned to some other function like a Re-configured Key or a User Routine. The key you enter will replace the '?' prompt and an '=' sign will be added, eg.,

STRING ! =

Then enter a string or else type RETURN if you wish to undo a previously defined string. The string may be up to 32 characters. Use the RETURN key to terminate the string. If you wish to include a RETURN within the string, then type a 'SHIFTED RETURN'. This will cause a new-line on the screen, but will not terminate the string.

UP TO 32 STRINGS MAY BE ASSIGNED. A STRING WILL BE DISPLAYED WHEN ITS KEY IS PRESSED, BUT ONLY IN DIRECT MODE. IN RUN MODE, THE VALUE OF THE ORIGINAL KEY, NOT THE STRING, WILL BE PASSED TO THE PROGRAM.

Re-Configured Keys

You may re-configure a key or "undo" a previously re-configured key by entering the command:

CONTROL and CURSOR RIGHT

Key Wizard will display the message:

CONFIG ?

Once again, type RETURN to abort. Otherwise enter the key you want to re-configure. Just like Strings, all values are permissible and similar restrictions apply. Hit RETURN to undo the key (ie if it was previously re-configured) or else enter the new character which will take its place. When you assign a value, use the ORIGINAL Key, not any other which may have been re-configured to this value. For example, if you assign "A" to key "Q" and then wish to assign "Q" to key "J", use the "Q" key for the assignment, even though the "Q" key has already been given a value of "A" In this way you may rearrange as many keys as you wish.

RE-CONFIGURED KEYS MAINTAIN THEIR ASSIGNED VALUES IN BOTH DIRECT AND RUN MODES.

User Routine Keys

You may assign a User Routine to a key or undo a previously assigned routine by entering the command:

CONTROL and RUN STOP

Key Wizard will display the message:

USER ?

Use RETURN to abort, otherwise enter the key you wish to have assigned or to undo. Hit RETURN to undo an assignment or else enter the start address of the User Routine in Hexadecimal. A 4 character address is mandatory. The assigned routine will be automatically started whenever the key is pressed.

You should bear in mind that the User Routine runs within the IRQ Interrupt. This facility allows you to develop your own editing, or other suitable, routines and then plug them into Key Wizard. Return to Key Wizard by using an "RTS" instruction with register "X" containing any standard ASCII keyboard code you may wish to have displayed or actioned. If you do not wish to pass back an ASCII value then set "X" to the value, $FF.

Background Routine

You may wish to run a background task of your own on every keyboard (timer) interrupt. To do this, change the SECOND and THIRD BYTES of the DATA Statement in LINE 200. I have used 35, 234, which is the address of an RTS instruction in the Kernel. By changing these to the address of your own routine, the JSR instruction (first byte, 32) will cause your own routine to be called. Remember the address is entered in the normal convention, low byte then high byte.

Conclusion

This ends Part 2 on the Commodore 64 Keyboard. I really enjoyed developing the Key Wizard system and hope you enjoy the result. Go on to Part 3 (next issue) if you have any interest in the design of a Keyboard Driver or if you wish to add to the routines already built into Key Wizard. You may find the exercise instructive and useful.

KEYWIZ-64: the BASIC loader/file saver for Key Wizard

```
IE  10 rem ********************************
HL  14 rem **  an amazing keyboard driver  **
DA  16 rem ****  for your commodore 64  ****
BO  18 rem ***  aubrey stanley  dec 1984  ***
CF  20 rem ********************************
KF  22 rem **
GI  50 tb$="kernel":pg$="kprg":rem *tells loader how
       to load kwiz*
FF  52 i=53216:nk=2:rem *irq address & n-key rollover
       count*
AG  54 if tb$="kernel" and pg$="kprg" then 100:rem *use
       kernel tables & data lines*
HA  56 if ft>1 then 100:rem *program and table files loaded
       from disk*
ME  58 if ft=1 then 62:rem *program file loaded from disk*
IN  60 ft=1:if pg$<>"kprg" then pg=1:load "0:"+pg$,8,1:
       rem *load program file*
GD  62 ft=2:if tb$<>"kernel" then tb=128:load "0:"
       +tb$,8,1:rem *load table file*
FN  64 goto 100
PH  65 rem ********************************
IK  70 rem *save program, type - run 70*
CB  72 pg$="kwiz":rem *program file name*
FP  74 ad=41216:bd=49152:n=2400:rem *prog address,
       saved from address, byte count*
PB  76 open 2,8,2,"@:"+pg$+",p,w"
IO  78 goto 88
```

```
JF  79 rem ***********
ID  80 rem *save tables, type - run 80*
ME  82 tb$="kwiz.tb":rem *table file name*
LO  84 ad=47072:bd=49152:n=2080:rem *table address,
       saved from address, byte count*
CC  86 open 2,8,2,"@:"+tb$+",p,w"
NK  88 print#2,chr$(ad-int(ad/256)*256);
BO  90 print#2,chr$(ad/256);
AE  92 for i=0 to n-1
JO  94 print#2,chr$(peek(bd+i));
PD  96 next i
AK  98 close 2:end
BK  99 rem ******************************
EF 100 ch=0:for n=0 to 23:rem *irq vector code*
NF 102 read a:poke i+n,a:ch=ch+a:next
ML 103 if ch<>1996 then print" error in data":end
DN 104 if pg=1 then 112:rem *bypass program pokes if
       program loaded from disk*
FC 106 ch=0:for j=41216 to 43615:rem *kwiz program
       code*
LE 108 read a:poke j,a:ch=ch+a:next
LL 109 if ch<>301952 then print" error in data":end
LP 112 poke 41216,tb:rem *tells program whether tables are
       loaded from disk*
AF 116 poke 41218,nk:rem *tells program the n-key factor*
HF 118 poke 56334,peek(56334)and254:rem *disable
       keyboard (timer) interrupt*
GH 120 poke 788,(i-int(i/256)*256):rem *set up irq vector
       address low byte*
CG 122 poke 789,(i/256):rem *set up irq vector high byte*
KM 124 poke 56334,peek(56334)or1:rem *enable keyboard
       (timer) interrupt
NE 126 end:rem *of loading*
KG 128 rem ******************************
NP 200 data32,35,234,165,1,41,254,
       133,1,32,39,161,165,1,9,1,133
LD 201 data1,76,126,234,0,90,32
EO 250 rem ******************************
AB 300 data   0,   0,   2,   0, 184,   0, 185,   0, 186,   0
NG 301 data 187, 254, 253, 251, 247, 239, 223, 191, 127, 255
NE 302 data 255, 255, 255, 255, 255, 255, 255,   0,   0,   0
BD 303 data   0,   0,   0,   0,   0,   0,   0,   0,   0,  44
AF 304 data   1, 161,  48,  62,  32, 185, 163, 169, 128, 141
OK 305 data   1, 161, 169,  46, 160, 165, 141, 143,   2, 140
IC 306 data 144,   2, 169,   0, 133, 198, 141, 141,   2, 141
DN 307 data 142,   2, 141,  27, 161, 141,  30, 161, 141,  37
CG 308 data 161, 141,  38, 161, 169,  64, 133, 203, 133, 197
HE 309 data 141,  28, 161, 141,  29, 161, 169, 255, 162,   7
KH 310 data 157,  19, 161, 202,  16, 250,  32, 110, 163, 173
ME 311 data  38, 161, 208,  17, 173,  30, 161, 208,  20, 169
FN 312 data   0, 141,   0, 220, 173,   1, 220, 201, 255, 208
GE 313 data   8, 162, 254,  32,  55, 163,  76, 150, 161, 169
BA 314 data   8, 141,  31, 161,  32, 156, 161,  32, 207, 162
OO 315 data 169, 127, 141,   0, 220,  96, 206,  31, 161,  16
HK 316 data   1,  96, 174,  31, 161, 189,  11, 161, 141,   0
PN 317 data 220, 173,   1, 220, 205,   1, 220, 208, 248,  93
DH 318 data  19, 161, 141,  32, 161, 172,  32, 161, 240, 222
EG 319 data 174,  31, 161, 136, 152,  45,  32, 161, 168,  77
OP 320 data  32, 161, 141,  33, 161, 140,  32, 161,  93,  19
OP 321 data 161,  45,  33, 161, 141,  34, 161, 240,  15, 206
OG 322 data  30, 161, 173,  33, 161,  93,  19, 161,  29,  19
AE 323 data 161,  76,   7, 162, 173,  30, 161, 205,   2, 161
HL 324 data 176, 199, 173, 141,   2,  41,   3, 201,   3, 240
HF 325 data 190, 238,  30, 161, 173,  33, 161,  93,  19, 161
DP 326 data  61,  19, 161, 157,  19, 161, 173,  31, 161,  10
HP 327 data  10,  10, 168, 173,  33, 161, 162, 255, 142,  36
ME 328 data 161, 238,  36, 161,  74, 144, 250, 152,  77,  36
KA 329 data 161, 141,  36, 161,  10,  10, 168, 173,   3, 161
LC 330 data 133, 245, 173,   4, 161, 133, 246, 177, 245, 170
BH 331 data 172,  36, 161, 224,   5, 176,  76, 224,   3, 240
PB 332 data  72,  77, 141,   2, 141, 141,   2, 141, 142,   2
NP 333 data  41,   4, 240,  34, 173, 141,   2,  41,   2, 240
PO 334 data  11, 169, 128,  77, 138,   2, 141, 138,   2,  76
JL 335 data 185, 161, 173, 141,   2,  41,   1, 240,  31,  77
EK 336 data  37, 161, 141,  37, 161,  76, 185, 161, 173, 141
ID 337 data   2,  41,   3, 201,   3, 208,  13, 173, 145,   2
IF 338 data  48,   8, 173,  24, 208,  73,   2, 141,  24, 208
LO 339 data  76, 185, 161, 173,  34, 161, 240,  29, 196, 197
KH 340 data 208,   6, 169,  64, 133, 203, 133, 197, 152, 160
BH 341 data   2, 136,  48,  10, 217,  28, 161, 208, 248, 169
PE 342 data  64, 153,  28, 161,  76, 185, 161,  32,  76, 163
JF 343 data 169,  64, 172,  27, 161, 173,  36, 161, 153,  28
EE 344 data 161, 152,  73,   1, 141,  27, 161, 169,  16, 141
PE 345 data 140,   2, 169,   4, 141, 139,   2,  32,  48, 163
EM 346 data  76, 185, 161, 173,  30, 161, 240,  91, 173, 141
EN 347 data   2,  41,   3, 201,   3, 240,  82,  44, 138,   2
GF 348 data  48,   2, 112,  75, 173, 140,   2, 240,   5, 206
PF 349 data 140,   2, 208,  65, 206, 139,   2, 208,  60, 169
LI 350 data   4, 141, 139,   2, 169,   2, 141,  35, 161, 206
GH 351 data  35, 161,  48,  45, 174,  35, 161, 189,  28, 161
IG 352 data 201,  64, 240, 241, 141,  36, 161,  32,  76, 163
JM 353 data  44, 138,   2,  48,  18,  41, 127, 201,  20, 240
AK 354 data  12, 201,  32, 240,   8, 201,  29, 240,   4, 201
OH 355 data  17, 208, 212,  32,  48, 163,  76, 253, 162,  96
ED 356 data 173,  36, 161, 133, 203, 133, 197, 108, 143,   2
HF 357 data 224, 224, 176,  13, 138, 166, 198, 236, 137,   2
HM 358 data 176,   5, 157, 119,   2, 230, 198,  96, 173,  36
IJ 359 data 161,  10,  10, 168, 173, 141,   2,  10, 201,   8
GG 360 data 144,   2, 169,   6, 170, 189,   3, 161, 133, 245
LK 361 data 189,   4, 161, 133, 246, 177, 245, 170, 172,  36
JM 362 data 161,  96,  32, 234, 255, 165, 204, 208,  41, 198
LK 363 data 205, 208,  37, 169,  20, 133, 205, 164, 211,  70
OL 364 data 207, 174, 135,   2, 177, 209, 176,  17, 230, 207
FD 365 data 133, 206,  32,  36, 234, 177, 243, 141, 135,   2
AE 366 data 174, 134,   2, 165, 206,  73, 128,  32,  28, 234
NJ 367 data 165,   1,  41,  16, 240,  10, 160,   0, 132, 192
EF 368 data 165,   1,   9,  32, 208,   8, 165, 192, 208,   6
AB 369 data 165,   1,  41,  31, 133,   1,  96, 173,   0, 161
JI 370 data 208,  44, 162,  63, 189, 129, 235, 157,   0, 160
FA 371 data 202,  16, 247, 162,  63, 189, 194, 235, 157,  64
OK 372 data 160, 202,  16, 247, 162,  63, 189,   3, 236, 157
AJ 373 data 128, 160, 202,  16, 247, 162,  63, 189, 120, 236
CK 374 data 157, 192, 160, 202,  16, 247, 173,   0, 161,  16
IJ 375 data   3,  76, 116, 164, 162, 228, 142, 199, 160, 232
JJ 376 data 142, 194, 160, 232, 142, 255, 160, 232, 142, 252
BK 377 data 160, 232, 142, 192, 160, 232, 142, 193, 160, 232
LG 378 data 142, 243, 160, 169,   0, 133, 245, 169, 184, 133
OA 379 data 246,  32, 117, 164, 189,   0, 160, 145, 245, 136
LP 380 data 136, 136, 136, 202,  16, 244, 169,   0, 133, 245
GO 381 data 169, 185, 133, 246,  32, 117, 164, 189,  64, 160
GH 382 data 145, 245, 136, 136, 136, 136, 202,  16, 244, 169
JK 383 data   0, 133, 245, 169, 186, 133, 246,  32, 117, 164
HI 384 data 189, 128, 160, 145, 245, 136, 136, 136, 136, 202
EA 385 data  16, 244, 169,   0, 133, 245, 169, 187, 133, 246
NO 386 data  32, 117, 164, 189, 192, 160, 145, 245, 136, 136
II 387 data 136, 136, 202,  16, 244, 162,  31, 138, 157, 224
DL 388 data 183, 202,  16, 249,  96, 160, 255, 169,   0, 145
LO 389 data 245, 136,  16, 251, 162,  63, 160, 252,  96, 252
EN 390 data 167,  79, 165,  78, 165, 235, 167, 105, 165, 119
DM 391 data 166,  16, 167,  43, 168, 120, 168, 158, 168, 139
MO 392 data 168,  78, 165,  78, 165,  78, 165,  78, 165,  78
KO 393 data 165,  78, 165,  78, 165,  78, 165,  78, 165,  78
LO 394 data 165,  78, 165,  78, 165,  78, 165,  78, 165,  78
MO 395 data 165,  78, 165,  78, 165,  78, 165,  78, 165,  78
OD 396 data 165,  78, 165,   0,   0, 188, 189, 190, 191,  83
GL 397 data  84,  82,  73,  78,  71,  32,  63, 157,  67,  79
FM 398 data  78,  71,  73,  71,  32,  63, 157,  85,  83,  69
JJ 399 data  82,  32,  63, 157,  32,  65,  76,  76,  32,  85
KE 400 data  83,  69,  68,  32,  85,  78,  68,  79,  78,  69
FB 401 data  32,  79,  46,  75,  46,  78,  85,  76,  76,  32
```

```
MC 402 data 75, 69, 89, 32, 65, 66, 79, 82, 84, 83
OA 403 data 65, 86, 69, 32, 80, 71, 83, 65, 86, 69
HM 404 data 32, 84, 66, 157, 95, 254, 226, 157, 255, 1
JM 405 data 255, 228, 0, 184, 8, 0, 185, 30, 224, 191
KJ 406 data 46, 165, 78, 165, 0, 0, 0, 0, 0, 0
PK 407 data 32, 52, 165, 76, 58, 163, 138, 224, 224, 176
FI 408 data 2, 169, 224, 41, 31, 10, 168, 185, 131, 164
JN 409 data 141, 195, 164, 185, 132, 164, 141, 196, 164, 108
LC 410 data 195, 164, 96, 32, 114, 169, 32, 188, 169, 174
FK 411 data 24, 165, 36, 157, 16, 242, 169, 78, 141, 38
HB 412 data 165, 169, 165, 141, 39, 165, 76, 239, 168, 169
OH 413 data 131, 141, 143, 2, 169, 165, 141, 144, 2, 169
PB 414 data 201, 133, 245, 169, 164, 133, 246, 169, 9, 32
CJ 415 data 226, 168, 76, 78, 170, 224, 254, 240, 110, 176
MF 416 data 8, 224, 227, 176, 107, 224, 13, 240, 113, 32
ND 417 data 114, 169, 32, 133, 169, 32, 188, 169, 173, 26
GL 418 data 165, 201, 226, 240, 87, 32, 152, 169, 169, 0
HB 419 data 141, 22, 165, 224, 225, 240, 47, 160, 32, 136
LJ 420 data 16, 16, 169, 226, 133, 245, 169, 164, 133, 246
CP 421 data 169, 9, 32, 226, 168, 76, 98, 166, 185, 224
EL 422 data 183, 201, 255, 240, 230, 141, 21, 165, 142, 20
ML 423 data 165, 169, 255, 153, 224, 183, 169, 225, 141, 19
PN 424 data 165, 32, 226, 169, 174, 20, 165, 224, 255, 208
DK 425 data 3, 32, 8, 170, 32, 58, 163, 169, 18, 141
NK 426 data 143, 2, 169, 166, 141, 144, 2, 162, 61, 76
CO 427 data 58, 163, 162, 63, 32, 58, 163, 162, 157, 76
HD 428 data 58, 163, 169, 255, 133
LH 500 data 245, 169, 164, 133, 246, 169, 6, 32, 226, 168
EF 501 data 76, 98, 166, 224, 254, 240, 41, 224, 227, 176
HO 502 data 222, 224, 225, 208, 9, 32, 114, 169, 32, 188
FA 503 data 169, 174, 24, 165, 224, 13, 240, 26, 173, 21
AD 504 data 165, 32, 207, 168, 172, 22, 165, 138, 145, 245
KC 505 data 200, 238, 22, 165, 192, 32, 240, 3, 76, 58
HP 506 data 163, 32, 58, 163, 173, 22, 165, 208, 18, 173
HE 507 data 21, 165, 168, 153, 224, 183, 173, 20, 165, 141
MF 508 data 19, 165, 32, 22, 170, 208, 3, 32, 36, 170
OC 509 data 32, 226, 169, 173, 40, 165, 141, 138, 2, 169
KM 510 data 46, 141, 143, 2, 169, 165, 141, 144, 2, 162
OA 511 data 141, 76, 58, 163, 169, 145, 141, 143, 2, 169
GO 512 data 166, 141, 144, 2, 169, 210, 133, 245, 169, 164
BB 513 data 133, 246, 169, 9, 32, 226, 168, 76, 78, 170
PP 514 data 224, 254, 240, 68, 176, 7, 224, 225, 144, 3
PM 515 data 76, 248, 165, 224, 13, 208, 3, 76, 2, 166
JB 516 data 32, 114, 169, 32, 133, 169, 32, 188, 169, 32
OJ 517 data 152, 169, 173, 22, 165, 201, 226, 240, 8, 142
NI 518 data 20, 165, 169, 226, 141, 22, 165, 174, 20, 165
MG 519 data 224, 255, 208, 3, 32, 8, 170, 32, 58, 163
KO 520 data 169, 220, 141, 143, 2, 169, 166, 141, 144, 2
GG 521 data 162, 61, 76, 58, 163, 224, 254, 240, 249, 224
KM 522 data 227, 144, 3, 76, 248, 165, 32, 114, 169, 32
EH 523 data 188, 169, 224, 225, 240, 7, 173, 26, 165, 201
BO 524 data 226, 208, 3, 174, 24, 165, 224, 13, 208, 8
GD 525 data 169, 0, 141, 22, 165, 76, 81, 166, 142, 19
NB 526 data 165, 32, 58, 163, 76, 92, 166, 169, 42, 141
HP 527 data 143, 2, 169, 167, 141, 144, 2, 169, 219, 133
KJ 528 data 245, 169, 164, 133, 246, 169, 7, 32, 226, 168
KP 529 data 76, 78, 170, 224, 254, 240, 82, 176, 18, 224
CA 530 data 224, 144, 7, 224, 227, 240, 10, 76, 248, 165
PH 531 data 224, 13, 208, 3, 76, 2, 166, 32, 114, 169
JJ 532 data 32, 133, 169, 32, 188, 169, 32, 152, 169, 224
CO 533 data 227, 240, 8, 142, 20, 165, 169, 227, 141, 19
HP 534 data 165, 169, 0, 141, 21, 165, 141, 22, 165, 174
KK 535 data 20, 165, 224, 255, 208, 3, 32, 8, 170, 32
EG 536 data 58, 163, 169, 0, 141, 41, 165, 169, 131, 141
PB 537 data 143, 2, 169, 167, 141, 144, 2, 162, 61, 76
EB 538 data 58, 163, 224, 254, 240, 249, 224, 228, 144, 3
HJ 539 data 76, 248, 165, 224, 225, 144, 9, 32, 114, 169
NC 540 data 32, 188, 169, 174, 24, 165, 224, 13, 208, 3
EP 541 data 76, 81, 166, 138, 201, 48, 144, 65, 201, 64
PE 542 data 240, 61, 144, 7, 201, 71, 176, 55, 24, 105
ID 543 data 9, 41, 15, 238, 41, 165, 172, 41, 165, 192
LD 544 data 4, 240, 8, 192, 2, 240, 4, 10, 10, 10
BB 545 data 10, 192, 3, 176, 9, 13, 22, 165, 141, 22
BE 546 data 165, 76, 223, 167, 13, 21, 165, 141, 21, 165
JE 547 data 192, 4, 240, 3, 76, 58, 163, 32, 58, 163
PA 548 data 76, 92, 166, 76, 248, 165, 32, 114, 169, 32
OE 549 data 188, 169, 174, 24, 165, 36, 157, 16, 3, 108
II 550 data 25, 165, 96, 36, 157, 16, 18, 169, 1, 45
KE 551 data 37, 161, 240, 11, 224, 20, 208, 8, 162, 29
ME 552 data 32, 58, 163, 162, 20, 96, 224, 32, 144, 19
CN 553 data 224, 128, 144, 4, 224, 160, 144, 11, 142, 20
IE 554 data 165, 162, 148, 32, 58, 163, 174, 20, 165, 96
HI 555 data 36, 157, 16, 72, 169, 183, 160, 224, 162, 192
KM 556 data 44, 42, 165, 48, 6, 169, 161, 160, 0, 162
NF 557 data 171, 133, 252, 132, 251, 169, 192, 133, 254, 160
HN 558 data 0, 132, 253, 177, 251, 145, 253, 200, 208, 249
AE 559 data 230, 252, 230, 254, 228, 252, 208, 241, 44, 42
PC 560 data 165, 48, 10, 169, 0, 141, 1, 192, 32, 50
FG 561 data 170, 208, 3, 32, 64, 170, 169, 128, 77, 42
CG 562 data 165, 141, 42, 165, 162, 141, 96, 36, 157, 16
IC 563 data 14, 172, 43, 165, 136, 16, 2, 160, 15, 140
DG 564 data 43, 165, 140, 33, 208, 96, 36, 157, 16, 14
II 565 data 172, 45, 165, 136, 16, 2, 160, 15, 140, 45
PC 566 data 165, 140, 32, 208, 96, 36, 157, 16, 42, 172
GH 567 data 44, 165, 136, 16, 2, 160, 15, 140, 44, 165
EN 568 data 140, 134, 2, 152, 160, 216, 132, 246, 160, 0
MA 569 data 132, 245, 145, 245, 200, 208, 4, 230, 246, 208
JE 570 data 247, 192, 232, 208, 243, 166, 246, 224, 219, 208
BF 571 data 237, 162, 255, 96, 168, 10, 10, 10, 10, 10
HL 572 data 133, 245, 152, 74, 74, 74, 168, 185, 197, 164
JD 573 data 133, 246, 96, 141, 41, 165, 160, 0, 32, 10
BI 574 data 169, 208, 251, 162, 255, 96, 173, 26, 165, 141
EA 575 data 41, 165, 173, 25, 165, 32, 207, 168, 160, 0
DH 576 data 76, 27, 169, 32, 10, 169, 208, 248, 162, 255
IA 577 data 108, 38, 165, 177, 245, 201, 141, 208, 2, 169
HB 578 data 13, 170, 32, 58, 163, 200, 206, 41, 165, 96
BC 579 data 165, 198, 205, 137, 2, 144, 222, 165, 245, 141
EN 580 data 34, 165, 165, 246, 141, 35, 165, 140, 33, 165
PN 581 data 173, 143, 2, 141, 36, 165, 173, 144, 2, 141
MJ 582 data 37, 165, 169, 77, 141, 143, 2, 169, 169, 141
LB 583 data 144, 2, 162, 255, 142, 38, 161, 76, 58, 163
OH 584 data 165, 198, 208, 244, 173, 34, 165, 133, 245, 173
IO 585 data 35, 165, 133, 246, 172, 33, 165, 173, 36, 165
DA 586 data 141, 143, 2, 173, 37, 165, 141, 144, 2, 169
NH 587 data 0, 141, 38, 161, 76, 0, 169, 173, 36, 161
DP 588 data 10, 10, 141, 30, 165, 165, 245, 141, 31, 165
BM 589 data 165, 246, 141, 32, 165, 96, 173, 30, 165, 141
HF 590 data 27, 165, 173, 31, 165, 141, 28, 165, 173, 32
OG 591 data 165, 141, 29, 165, 96, 173, 23, 165, 141, 19
IM 592 data 165, 173, 24, 165, 141, 20, 165, 173, 25, 165
GG 593 data 141, 21, 165, 173, 26, 165, 141, 22, 165, 96
BL 594 data 173, 28, 165, 133, 245, 173, 29, 165, 133, 246
NK 595 data 96, 173, 31, 165, 133, 245, 173, 32, 165, 133
HM 596 data 246, 173, 30, 165, 168, 177, 245, 141, 23, 165
NF 597 data 200, 177, 245, 141, 24, 165, 200, 177, 245, 141
PP 598 data 25, 165, 200, 177, 245, 141, 26, 165, 96, 173
EK 599 data 28, 165, 133, 245, 173, 29, 165, 133, 246, 173
DJ 600 data 27, 165, 168, 173, 19, 165, 145, 245, 200, 173
GH 601 data 20, 165, 145, 245, 200, 173, 21, 165, 145, 245
PJ 602 data 200, 173, 22, 165, 145, 245, 96, 169, 247, 133
IO 603 data 245, 169, 164, 133, 246, 169, 8, 32, 226, 168
IA 604 data 96, 169, 235, 133, 245, 169, 164, 133, 246, 169
KI 605 data 7, 32, 226, 168, 96, 169, 242, 133, 245, 169
NI 606 data 164, 133, 246, 169, 5, 32, 226, 168, 96, 169
FB 607 data 5, 133, 245, 169, 165, 133, 246, 169, 7, 32
FJ 608 data 226, 168, 96, 169, 12, 133, 245, 169, 165, 133
AJ 609 data 246, 169, 7, 32, 226, 168, 96, 173, 138, 2
CB 610 data 141, 40, 165, 169, 64, 141, 138, 2, 96, 1
GL 611 data 255, 1, 255, 1, 255
```

Linked Lists Part 2

K. Murray Smith
London, Ont.

In Part 1 we examined how linked lists could be used to "sort" the records in a file into a particular order. It was shown that the technique of linking avoided the shuffling of elements within the storage array, allowed all the items to be kept in the same array and permitted different groups of these items to be sorted on different fields.

The Rentawreck rental agency now has all of its vehicles linked into three lists: one for those rented (linked by due date), another for those available (linked by mileage), and a final list for the vehicles currently being serviced (also linked by mileage).

We now must write a program which will let the company update its lists as vehicles are returned, rented, serviced, sold or purchased, and also display or print any one of the lists in *correct order*. This program will be one to be used interactively over and over again, whereas the previous startup program was only used once to produce the initial linkages. For ease of use this program will be menu-driven. Figure 1 shows the menu choices.

Loading the Lists

The linked lists generated in Part 1 were stored in a sequential file along with the size of the arrays being used and the entry points to each of the linked lists. Program 1 reads these data. Table 1 defines the variables used.
After reading in the array size (S) which had been used for saving the file (line 30), the user is then asked if he/she wishes to increase the array sizes (lines 70–140). This might be required when several new vehicles are purchased. As an assist, the number of elements currently used is shown (40–50). Following this decision, the stored data is read in (240–270). If the arrays were increased in size the extra elements in the licence plate, due date and mileage arrays are now set to blanks (320–340) as in the Part 1 program and the pointer list altered to accommodate the new free elements (360–410).

Lines 230 and 260 are needed due to the dynamic nature of BASIC strings. This provides for very flexible programming as it is not necessary to tell the computer how many bytes to allocate for each string variable; rather the allocation is based on the number of bytes actually used. An unfortunate side effect of this is that when a string such as " ABC" is saved in a sequential file and then recalled, the leading blanks are lost. Since the decision was made in Part 1 to use strings to store mileages, the values read from the file must be "massaged" to put in leading blanks where necessary. Otherwise the mileage string "1569" will be *greater* than the mileage string "14583"! Lines 230 and 260 make sure "1569" is used as " 1569".

Displaying or Printing a List

To display or print one of the lists in order we simply enter the array using the pointer to the start of the list and then, using the linkage array, follow the list through until a zero is found as a pointer to the next element. Menu selection 0 or 1 will prompt with a second menu requesting which of the three lists is required.

Updating a List

Whenever a vehicle's status is altered, there are two lists affected. For example, when a vehicle is sold, pointers in the available portion of the link array must be altered to *ex*clude this vehicle and pointers for the free space must be altered to *in*clude the elements formerly used by this vehicle. When a rented vehicle is returned, pointers must be changed in both the rented and available sections of the link array.

Menu selections 2–7 control list updates to log a returned vehicle, to rent one, to send one for servicing, to remove a vehicle to be sold and to make available a vehicle that has just been purchased.

The Mechanics of Updating

Adding or removing an element to or from a list are the basic list updating operations. The classical way of deleting an element is to locate it in the array and move every element past it one location closer to the front. Figure 2a shows an alphabetic array from which D is to be deleted and Figure 2b shows the array after the deletion. Insertion of the letter H into the array of Figure 2b requires locating where the H belongs, moving every element past this spot one location farther back in the array and then inserting the H.

This method for insertion and deletion is not efficient if the operation occurs anywhere but close to the end of the array, if a multidimensional array is involved or if the elements of more than one array must be moved.

Linked lists show their superiority in the areas of adding and removing items as well. To delete an element, simply alter the pointers to and from this element and then return this element to the free list (making it the first one available for future use which will tend to keep early elements of arrays occupied). To insert a new quantity, it is first stored in the next free element, the pointer to the first free one is adjusted, its proper location is found and then the pointers to and from it are set. This seems like a great deal of work but the implementation is simple and it avoids shifting many array elements.

Changing an element from one linked list to another does not require a physical move of the data: one set of pointers is altered to exclude a specific element in one or more arrays and another set of pointers is altered to include this element.

To insert or delete an element at the beginning of a list requires that the variable storing the entry point to that linked list be updated appropriately. For elements at the end of a list, it must also be remembered to set the appropriate pointer to zero to indicate the end of a list.

The Updating Program

Program 2 contains *the additions* to Program 1 to perform the menu functions. In addition to outputting and updating lists, the menu provides an additional choice: a check for overdue vehicles in which all vehicles with a due date *prior* to the current date are displayed on the screen (with a printed-list option). In this way all the information on a possibly stolen vehicle is available quickly.

Table 2 contains a brief description of the new variables introduced in Program 2.

FL is used as a flag to indicate whether or not any lists have been altered during the running of the program. The flag is lowered (set to 0) in line 145 and will be raised (set to 1) if there are any changes.

Starting at line 500 the menu is presented (500–620), the selection is obtained (630) and checked for validity (640–660). Although all the choices are single digit, INPUT is used rather than GET because of the brief moment in which the user might detect an erroneous input and then be able to change it. As the program stands, this is important as it has not been "idiot-proofed" as would be necessary in a commercial package. Some of the options will not let you exit them until they have been successfully completed. Of course you could use the complementary routine to undo the error if this ever happens to you. Keep in mind as well that the purpose of the program is to illustrate the use of linked lists, not bullet-proofing!

Control is then transferred to the appropriate subroutine (670–700). Every return from a subroutine will send the user back to the main menu (690,710).

The block beginning with line 1000 handles options 0 and 1 for showing a particular list. A menu is presented (1010–1070) and the selection (1080) is checked for validity (1090–1110). If the option is not "3" to return to the main menu (1120), then the entry point to the appropriate list is established (1130–1150). If the entry point to a list happens to be a zero (1160), then there are no vehicles in the list. After opening a channel to the printer if necessary (1180), the information about the first vehicle in the list is displayed or printed as required (1190–1200) and a check is made to see if this is the last vehicle in the list (1210). If not, the variable EN is changed to point to the next item in the list (1220), this is printed or displayed and so on. An end of listing message is printed (1240), the printer channel is shut down if necessary (1250) and the program pauses (1260–1290) before returning to the display-print sub-menu.

The line-2000 block handles the return of a rented vehicle. After checking to see if in fact there are any vehicles rented (2020) and returning to the main menu if there are not (2030–2070), the returned vehicle's licence plate number and the new odometer reading are requested (2080–2150). The odometer reading is then massaged (2160–2170) as discussed previously and the vehicle is located in the rental list and removed (2190–2200). Since the subroutine at 11000 is called from several options in the main menu, the entry point for the rental list must be specified (2190). A value of 1 for F2 indicates that some problem occurred in the search subroutine and an immediate return to the main menu is performed (2210). Note that the value of FL is not changed until line 2290, that is until the vehicle has been successfully put into the available list (2250–2270). Having found the returned vehicle in the rented list, its mileage and due date entries are adjusted (2220–2230) and the pointer to the beginning of the rental list is updated (2240) if the first vehicle on the rented list was the one removed. Putting the returned vehicle in the available list also uses a call to a general subroutine at line 13000 and the correct list entry point is needed and this is also updated if necessary after the return (2250–2280).

The line-3000 block looks after the renting of a vehicle. It is very similar to the above block and so does not need to be described in detail. Line 3130 checks to see if there are more or fewer than 5 characters in the requested due date or if the month number is more than 12 or if the day is more than 31. Again not foolproof, but will trap quite a few common errors. Note also the different subroutine required to put the vehicle in the rented list (3250).

The blocks at lines 4000 and 5000 move the vehicles between the available list and the list of vehicles being serviced and are similar to the previous two blocks.

The line-6000 block controls the selling of a vehicle. In this operation the vehicle is only removed from a list and not added to another so there is no second GOSUB. Also the total number of vehicles owned by the company (or the number of array elements actually in use) is reduced by one (6180).

The next block makes a new vehicle available by finding the first available element (7200), storing this vehicle's data in this element of the plate number and mileage arrays (7220–7230) and then resetting the pointer to the start of the free space (7210). This vehicle is then linked into the available list (7240–7260) and the total number of vehicles is increased by one (7270).

The line-8000 block checks for overdue vehicles. The main part is the section which traces the path through the list of rented vehicles (8160–8220).

The final block of coding coming from the main menu begins at line 9000. If the flag FL is still down, then no changes have been made to any lists and so the program stops at this point (9010). If the flag is raised (value 1), then the old file called LINKED LISTS is scratched (9020–9040) and an updated one of the same name is created (9050–9120). Line 9050 defines a carriage return variable for use in the PRINT# operation.

Blocks beginning with lines 11000 and 12000 locate a vehicle by plate number and remove it from that list. Since they differ only in the last couple of lines, it is possible to combine them if a check is made to see which finishing part is required and if the calling routines use a pointer which can be passed to the subroutine to indicate which part is needed. However, the increased program complexity needed to save less than a dozen lines is not warranted.

At the beginning of the line-11000 block a flag is set low (11010) and it will be raised (set to 1) if the vehicle is not found in the list. The entry point to the appropriate list is copied to K (11020). A check is made to see if the plate number being looked for is at the beginning of the list (11030). If not, this subscript is stored for future use (11040) and the element to be examined is set to the next one in the list (11050). If this element is not zero (11060), then it is checked for a match. If the element number is zero, we are at the end of the list with no match made. A message is printed, the flag F2 is raised and we return to the calling routine (11070–11120). If a match was found in line 11030, then a check is made to see if the match occurred for the first element in the list (11130). If so, this vehicle is removed from the list by setting the entry point to this list to the next element of the list (11140). If not, the element in the link array which pointed to *this* element is set to skip this element and point to the *next* one (11160). In either of the

above cases, the location of the match is kept for future use (11170).

As mentioned before, the line–12000 block is almost the same as the one described above. However, when a vehicle is sold we do not have to remember where it is in the array. In fact, we want to make that location available for use if a vehicle is purchased and so we set the pointer of the removed vehicle to point to the first element of the free list (12160) and then make the location of the removed vehicle the new start of the free list (12180).

The blocks at lines 13000 and 14000 locate by mileage and due date respectively the correct location for an item in a list and then insert the item in that spot. As discussed previously, these routines also could be combined into one. As in the above two blocks, the list is searched to find the correct location for the vehicle whose data are in the elements numbered N of the arrays (13010–13050 or 14010–14050). This N is the quantity that had been saved above for future use. If the vehicle belongs at the end of the list, the pointer of the last element in the list is set to point to element N (13060 or 14060) and the pointer for element N is set to zero to indicate the end of the list (13070 or 14070). If it belongs at the beginning of the list, then the list entry value is set to N (13100 or 14100) and the pointer for element N is set to what used to be the first element of the list (13130 or 14130). If the vehicle belongs somewhere between the endpoints of the list, the pointer of the previous vehicle is changed to point to this one (13120 or 14120) and this one's pointer is set to the next vehicle's element number (13130 or 14130).

Summary

Now that the updating program for Rentawreck is debugged and working, the sample DATA statements used in the Part 1 program can be replaced by the actual vehicle data and the array sizes increased to a sufficiently large value. More arrays for additional fields of information can also be added now.

If you have had an opportunity to run the final programs from Part 1 and Part 2, you may be disappointed at how long it takes to establish the initial linked lists and surprised at how rapidly you return to the main menu after making a change in a list. Remember though that the initial setup is done only once. (A machine language subroutine for the actual linking section might appeal to someone out there as well.)

There are several features of linked lists which have not been addressed in the car rental example. Consider the request for all of the data about a vehicle whose licence number you are given. It is awkward to have to go through all three lists looking for this vehicle. If all the vehicles were linked alphabetically by licence plate, the search would be faster. The programming cost to implement this? One more array.

Also note that all of the searches begin at the start of the linked list. In the case of a search of all vehicles for a particular plate number, why not just look at the first entry in the plate array? If this is before the plate we want then go on further following the linkage, but if it is past the one we want then why not back up through the list? To do this would require what is known as backward linking of the plate numbers, a fairly easy task once forward linking is understood.

Possibly by now you have thought of some other applications for linked lists – that coin or stamp collection you have always meant to organize, your recipes, that mailing list which seems to take forever to update and re–sort. You may even have seen that some of the sections of the Part 2 program can be easily written in machine code. Once you have the ability to add items to or remove them from a linked list and you realize that single characters entered from the keyboard could be the items in a linked list, then you might be able to write that special routine you seem to need all the time but can't find in a wordprocessor within the reach of your budget. . .

Figure 1
LINKED LIST MENU

```
Display a list.....0
Print a list.....1
Return a vehicle.....2
Rent a vehicle.....3
Remove a vehicle for service.....4
Make serviced vehicle available.....5
Sell a vehicle.....6
Add a new vehicle.....7
Check for overdue vehicles.....8
Quit program.....9
```

Figure 2

```
2a  A B D G I  K M   before deletion of D
2b  A B G I  K M __  after deletion of D
2c  A B G H I  K M   after insertion of H
```

Table 1
Description of Variables

B$	a string of 6 blanks to provide leading blanks where necessary in the mileage strings
D$	single letter answer to any yes/no question
LD	the array of pointers for the linked lists
NS	new array size if needed
PL$,DU$,MI$	arrays for licence plate nums, due dates, mileages
S	the size of the arrays stored in the file
SZ	the size to which arrays will be dimensioned
SR,SA,SS	the element numbers for the start of the lists of vehicles currently rented, available or being serviced
SF	the first free element in the arrays
UL	the number of elements actually used in arrays in the file

Table 2
Description of Variables

C1	main menu choice
C2	display–print menu choice
CR$	holds a keyboard entry (expecting a carriage return)
DA$	today's date
EN	current element number while going through a linked list
FL	a flag to indicate whether changes have been made to any list
F2	a flag to indicate that an abnormal condition caused an exit from a GOSUB
K	the current element being examined in a list (similar to EN)
LS	subscript of the last element examined in a list
N	location of an element to be added to a list
ND$	an input due date
NM$	an input odometer reading
NP$	an input licence plate number
S$	a blank space used for output
SE	entry point to a linked list

Program 1

```
KG  10 rem- program 1
AJ  20 open1,8,2," 0:linked lists,r "
MB  30 input#1,s,ul,sr,sa,ss,sf
EI  40 print chr$(147)" the array sizes are ";s
ME  50 print ul;" elements are currently used. "
MD  60 for i = 1 to 1000:next i
EG  70 print " do you wish to increase the array sizes "
HB  80 print " at this time? "
AL  90 print " (type y or n): ";
CA 100 get d$
JP 110 if d$ = "" then 100
GA 120 print d$
KL 130 if d$ = " n " then 210
IL 140 if d$<>" y " then 90
LI 150 input " new array size ";ns
FJ 160 if ns>s then 190
NH 170 print " new size must be greater than ";s
NF 180 goto 150
OC 190 sz = ns
IG 200 goto 220
BK 210 sz = s
JK 220 dim pl$(sz),du$(sz),mi$(sz),ld(sz)
EA 230 b$ = " [6 spaces] "
HA 240 for i = 1 to s
HA 250 :input#1,pl$(i),du$(i),mi$(i),ld(i)
KJ 260 mi$(i) = left$(b$,6-len(mi$(i))) + mi$(i)
NO 270 next i
LO 280 close1
JN 290 if ns = 0 then 440
ED 300 rem- initialize new array sections
JJ 310 for i = s + 1 to ns
DC 320 :pl$(i) = " [7 spaces] "
NC 330 :du$(i) = " [5 spaces] "
CC 340 :mi$(i) = " [6 spaces] "
ND 350 next i
GP 360 rem- link rest of free space
GG 370 i = s
BG 380 ld(i) = i + 1
NM 390 :i = i + 1
KJ 400 if i<ns then 380
FA 410 ld(ns) = 0
CL 420 rem- make array size change permanent
DC 430 if sz>s then s = sz
EO 440 print "    << list loading completed >> "
PB 19999 end
```

Program 2

```
NG  10 rem- program 2
JP 145 fl = 0
OJ 500 rem- main menu
AL 510 print " S]]]]] rentawreck car rental agency "
BH 520 printtab(3);" q display a list . . . . . .0 "
OI 530 printtab(3);" q print a list . . . . . .1 "
HL 540 printtab(3);" q return a vehicle . . . . . .2 "
BE 550 printtab(3);" q rent a vehicle . . . . . .3 "
GJ 560 printtab(3);" q remove a vehicle for service . . . .4 "
CO 570 printtab(3);" q make serviced vehicle available. .5 "
GE 580 printtab(3);" q sell a vehicle. . . . . .6 "
LD 590 printtab(3);" q add a new vehicle . . . . . .7 "
CD 600 printtab(3);" q check for overdue vehicles . . . . .8 "
BK 610 printtab(3);" q quit program . . . . . .9 "
CP 620 printtab(12);" q your choice (0-9) ";"  ]]]] ";
NA 630 input c1
PD 640 if c1>=0 and c1<=9 then 670
OA 650 print " <<invalid choice: must be 0-9>> QQQ "
AE 660 goto 620
FL 670 if c1>1 then 700
FK 680 gosub 1000
DF 690 goto 500
HH 700 on c1-1 gosub 2000,3000,4000,5000,
        6000,7000,8000,9000
HG 710 goto 500
MH 1000 rem- display (c1 = 0) or print (c1 = 1)a list
HE 1010 if c1 = 0 then print " Sqq ";tab(14);" display menu "
EG 1020 if c1 = 1 then print " Sqq ";tab(14);" print menu "
PM 1030 printtab(8);" q rented vehicles . . . . . .0 "
GE 1040 printtab(8);" q available vehicles . . . . . .1 "
AD 1050 printtab(8);" q vehicles being serviced. . .2 "
PH 1060 printtab(8);" q return to main menu . . . . .3 "
CK 1070 printtab(12);" q your choice (0-3) ";"  ]]]] ";
DN 1080 input c2
MD 1090 if c2>=0 and c2<=3 then 1120
IL 1100 print " <<invalid choice: must be 0-3>> QQQ "
CF 1110 goto 1070
ME 1120 if c2 = 3 then return
PB 1130 if c2 = 0 then en = sr
HO 1140 if c2 = 1 then en = sa
LD 1150 if c2 = 2 then en = ss
KJ 1160 if en = 0 then 1240
CN 1170 s$ = "  "
JO 1180 if c1 = 1 then open1,4
LF 1190 if c1 = 0 then print pl$(en);s$;du$(en);s$;mi$(en)
KB 1200 if c1 = 1 then print#1, pl$(en);s$;du$(en);s$;mi$(en)
MM 1210 if en = 0 then 1240
AA 1220 en = ld(en)
AN 1230 goto 1190
EE 1240 printtab(11);" qq << end of listing >> "
PD 1250 if c1 = 1 then close1
GK 1260 print " q press <return> to return to menu "
NG 1270 get cr$
CF 1280 if cr$<>chr$(13) then 1270
PP 1290 goto 1000
PO 2000 rem- return a vehicle
CM 2010 printtab(5);" Sqq return a vehicle... "
JD 2020 if sr>0 then 2080
IN 2030 print " q no vehicles currently rented... "
DK 2040 print " q press <return> to return to main menu "
JH 2050 get cr$
CF 2060 if cr$<>chr$(13) then 2050
CD 2070 return
KC 2080 input " plate number ";np$
JC 2090 if len(np$) = 7 then 2120
GA 2100 print " **plate must be 7 characters...** "
OD 2110 goto 2080
GC 2120 input " odometer reading ";nm$
EJ 2130 if len(nm$)<=6 then 2160
JP 2140 print " q **reading too large...999999 max ** "
EG 2150 goto 2120
OI 2160 b$ = " [6 spaces] "
HD 2170 nm$ = left$(b$,6-len(nm$)) + nm$
BE 2180 rem- locate vehicle in rental list and remove
CM 2190 se = sr
JC 2200 gosub 11000
DJ 2210 if f2 = 1 then return
MA 2220 mi$(n) = nm$
```

```
KD  2230 du$(n) = " [5 spaces] "
BA  2240 sr = se
BC  2250 rem- put vehicle in available list
GO  2260 se = sa
HH  2270 gosub 13000
GP  2280 sa = se
LF  2290 fl = 1
IB  2300 return
MA  3000 rem- rent a vehicle
GJ  3010 printtab(5); " Sqq rent a vehicle... "
BO  3020 if sa>0 then 3080
DM  3030 print " q no vehicles currently available... "
LI  3040 print " q press <return> to return to main menu "
BG  3050 get cr$
MD  3060 if cr$<>chr$(13) then 3050
KB  3070 return
CB  3080 input " plate number " ;np$
FB  3090 if len(np$) = 7 then 3120
OO  3100 print " **plate must be 7 characters...** "
JC  3110 goto 3080
DE  3120 input " due date (as mm-dd) " ;nd$
OO  3130 if len(nd$)< = 5 and left$(nd$,2)< = " 12 "
         and right$(nd$,2)< = " 31 " then 3160
KC  3140 print " q **invalid due date** "
PE  3150 goto 3120
FB  3160 rem- locate vehicle in rental list and remove
EH  3170 se = sa
NP  3180 gosub 11000
HG  3190 if f2 = 1 then return
AO  3200 du$(n) = nd$
MP  3210 mi$(n) = " [6 spaces] "
CK  3220 sa = se
KL  3230 rem- put vehicle in rented list
MN  3240 se = sr
PE  3250 gosub 14000
NP  3260 sr = se
PC  3270 fl = 1
MO  3280 return
AD  4000 rem- remove a vehicle for service
PC  4010 printtab(5); " Sqq send a vehicle for servicing... "
NM  4020 if sa>0 then 4080
LK  4030 print " q no vehicles currently available... "
DH  4040 print " q press <return> to return to main menu "
JE  4050 get cr$
GC  4060 if cr$<>chr$(13) then 4050
CA  4070 return
KP  4080 input " plate number " ;np$
BA  4090 if len(np$) = 7 then 4120
GN  4100 print " **plate must be 7 characters...** "
EB  4110 goto 4080
AO  4120 rem- locate vehicle in available list and remove
ED  4130 se = sa
NL  4140 gosub 11000
HC  4150 if f2 = 1 then return
OE  4160 sa = se
BA  4170 rem- put vehicle in service list
KI  4180 se = ss
HG  4190 nm$ = mi$(n)
BA  4200 gosub 13000
GL  4210 ss = se
FO  4220 fl = 1
CK  4230 return
OA  5000 rem- make a serviced vehicle available
AE  5010 printtab(5); " Sqq return a serviced vehicle... "
BA  5020 if ss>0 then 5080
CH  5030 print " q no vehicles being serviced... "
LF  5040 print " q press <return> to return to main menu "
BD  5050 get cr$
AB  5060 if cr$<>chr$(13) then 5050
KO  5070 return
CO  5080 input " plate number " ;np$
NO  5090 if len(np$) = 7 then 5120
OL  5100 print " **plate must be 7 characters...** "
PP  5110 goto 5080
IM  5120 rem- locate vehicle in available list and remove
AE  5130 se = ss
FK  5140 gosub 11000
PA  5150 if f2 = 1 then return
MG  5160 ss = se
JO  5170 rem- put vehicle in service list
OE  5180 se = sa
PE  5190 nm$ = mi$(n)
JO  5200 gosub 13000
IG  5210 sa = se
NM  5220 fl = 1
KI  5230 return
FF  6000 rem- to sell a vehicle
PD  6010 printtab(5); " Sqq sell a vehicle... "
HK  6020 if sa>0 then 6090
FM  6030 print " q available list empty... "
JN  6040 print " vehicle must be here to be sold "
NE  6050 print " q press <return> to return to main menu "
DC  6060 get cr$
IA  6070 if cr$<>chr$(13) then 6060
MN  6080 return
EN  6090 input " plate number " ;np$
FO  6100 if len(np$) = 7 then 6130
AL  6110 print " **plate must be 7 characters...** "
FP  6120 goto 6090
AC  6130 rem- search for vehicle and remove
OA  6140 se = sa
LJ  6150 gosub 12000
BA  6160 if f2 = 1 then return
IC  6170 sa = se
MC  6180 ul = ul-1
HJ  6190 fl = 1
EF  6200 return
JO  7000 rem- add a new vehicle
BM  7010 printtab(5); " Sqq add a new vehicle... "
HK  7020 if sf>0 then 7090
JC  7030 print " q array full - no free space "
OO  7040 print " addition cannot be made "
FD  7050 print " q press <return> to return to main menu "
LA  7060 get cr$
CP  7070 if cr$<>chr$(13) then 7060
EM  7080 return
ML  7090 input " plate number " ;np$
BN  7100 if len(np$) = 7 then 7130
IJ  7110 print " **plate must be 7 characters...** "
AO  7120 goto 7090
IL  7130 input " odometer reading " ;nm$
OC  7140 if len(nm$)< = 6 then 7170
LI  7150 print " q **reading too large...999999 max ** "
GA  7160 goto 7130
AC  7170 b$ = " [6 spaces] "
JM  7180 nm$ = left$(b$,6-len(nm$)) + nm$
JB  7190 rem- add vehicle to available list
EG  7200 n = sf
```

```
KJ  7210 sf=ld(sf)
JK  7220 pl$(n)=np$
OJ  7230 mi$(n)=nm$
KF  7240 se=sa
LO  7250 gosub 13000
KG  7260 sa=se
LG  7270 ul=ul+1
JN  7280 fl=1
GJ  7290 return
HG  8000 rem- check for overdue vehicles
GN  8010 if sr<>0 then 8040
OI  8020 print" q there are no vehicles currently rented"
KH  8030 return
LI  8040 print" Sq would you like a printed list also?"
IM  8050 print"(type y or n):";
KB  8060 get d$
KO  8070 if d$="" then 8060
OB  8080 printd$
CB  8090 if d$="n" then 8110
CG  8100 if d$<>"y" then 8050
JE  8110 print"enter today's date as mm-dd <return>"
OI  8120 input "date";da$
OC  8130 if len(da$)<=5 and left$(da$,2)<="12"
         and right$(da$,2)<="31" then 8160
CL  8140 print" q **invalid due date**"
GO  8150 goto 8120
DB  8160 en=sr
BE  8170 if d$="y" then open1,4
IN  8180 if du$(en)>=da$ then 8210
CA  8190 printpl$(en);" ";du$(en);" ";mi$(en);" is overdue"
PN  8200 if d$="y" then print#1,pl$(en);" ";du$(en);
         " ";mi$(en);" is overdue"
PJ  8210 :en=ld(en)
MH  8220 if en<>0 then 8180
IE  8230 printtab(11);" q<< end of listing >>"
FK  8240 if d$="y" then close1
FO  8250 print" q press <return> to return to main menu"
LL  8260 get cr$
KK  8270 if cr$<>chr$(13) then 8260
EH  8280 return
BP  9000 rem- quit program, saving lists if necessary
PE  9010 if fl=0 then 19999
EL  9020 open1,8,15
OD  9030 print#1,"s0:linked lists"
DC  9040 close1
DL  9050 c$=chr$(13)
LK  9060 open1,8,2,"0:linked lists,seq,w"
OE  9070 print#1,s;c$;ul;c$;sr;c$;sa;c$;ss;c$;sf
PI  9080 for i=1 to s
AE  9090 :print#1,pl$(i);c$;du$(i);c$;mi$(i);c$;ld(i)
LG  9100 next i
JG  9110 close1
NE  9120 printtab(10);"lists have been saved"
JI  9130 goto 19999
OJ  9140 rem******************************
CA  11000 rem- locate and remove a vehicle
MB  11010 f2=0
JE  11020 k=se
KO  11030 if np$=pl$(k) then 11130
EM  11040 ls=k
PB  11050 k=ld(k)
LK  11060 if k<>0 then 11030
BL  11070 print" q plate not found in list"
DP  11080 print" q press <return> to return to main menu"
JM  11090 get cr$
IP  11100 if cr$<>chr$(13) then 11090
BI  11110 f2=1
MI  11120 return
NC  11130 if k<>se then 11160
AP  11140 se=ld(se)
GC  11150 goto 11170
CH  11160 ld(ls)=ld(k)
AI  11170 n=k
IM  11180 return
JO  12000 rem- locate and remove a sold vehicle
EA  12010 f2=0
BD  12020 k=se
FN  12030 if np$=pl$(k) then 12130
MK  12040 ls=k
HA  12050 k=ld(k)
EJ  12060 if k<>0 then 12030
JJ  12070 print" q plate not found in list"
LN  12080 print" q press <return> to return to main menu"
BL  12090 get cr$
DO  12100 if cr$<>chr$(13) then 12090
JG  12110 f2=1
EH  12120 return
HB  12130 if k<>se then 12160
IN  12140 se=ld(se)
CB  12150 goto 12170
KF  12160 ld(ls)=ld(k)
DN  12170 ld(k)=sf
PB  12180 sf=k
KL  12190 return
JA  13000 rem- locate by mileage the proper location in list
          for a new element
PA  13010 k=se
BC  13020 if nm$<=mi$(k) then 13090
CN  13030 :ls=k
OK  13040 :k=ld(k)
AH  13050 if k<>0 then 13020
LI  13060 ld(ls)=n
II  13070 ld(n)=0
ED  13080 return
JM  13090 if k<>se then 13120
HL  13100 se=n
OM  13110 goto 13130
HM  13120 ld(ls)=n
AD  13130 ld(n)=k
AH  13140 return
FN  14000 rem- locate by due date the proper location in
          list for a new element
HP  14010 k=se
BN  14020 if nd$<=du$(k) then 14090
KL  14030 :ls=k
GJ  14040 :k=ld(k)
JF  14050 if k<>0 then 14020
DH  14060 ld(ls)=n
AH  14070 ld(n)=0
MB  14080 return
DL  14090 if k<>se then 14120
PJ  14100 se=n
KL  14110 goto 14130
PK  14120 ld(ls)=n
IB  14130 ld(n)=k
IF  14140 return
```

A High Resolution Graphics Utility For The 64

Gary Kiziak
Burlington, Ont.

There is no question regarding the superb graphic capabilities of the Commodore 64 – just look at the wealth of fine educational software and the unbelievable games that are now available. Yet most people are simply not able to make use of these capabilities when writing their own programs. Why is this?

Actually, the answer is quite simple. Most people write their programs in BASIC, and unfortunately there are no commands that allow you access to these graphic capabilities other than PEEK and POKE. Of course there are some very good extensions to the language such as SIMON'S BASIC, the SUPER EXPANDER cartridge, and many more. These extensions provide additional commands that allow you to draw straight lines and circles, and to access the other graphics capabilities in a very simple way. The only drawback to using one of these extensions will arise if you plan to share any of your programs with others. For unless they have the same graphics package as you, they will not be able to make use of your programs.

In this article, I will present a series of commands that will allow you to access some of the hires graphics capabilities of the 64 in a simple way. These commands will not be as comprehensive as the above-mentioned extensions, but they will be certainly more than adequate. Best of all, you can share it along with any of your own programs that make use of it.

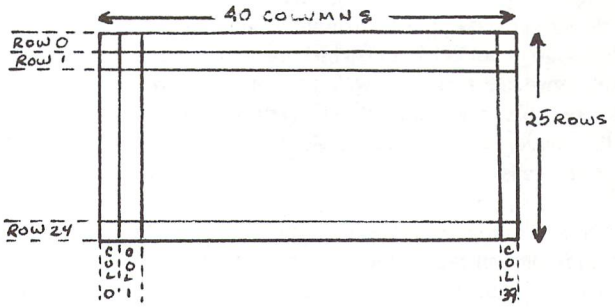
The commands will allow you to plot points, draw lines and boxes, and with a little help from BASIC draw circles, ellipses, etc. These drawings can take place on the normal hires screen or the more colourful multi-colour screen. You will even be able to print onto the hires screen, so that all your elaborate creations can be labelled using the built in character sets or even using custom character sets that you created.

Some Preliminary Information

The make-up of the normal text screen should be well known, but for completeness let's go through some of the details.

The text screen is made up of 25 rows, each containing 40 characters. Each of these 1000 (i.e. 25*40) 'character cells' can display a character in any of 16 different colours. Within a single character cell however, only two colours can be displayed – the foreground or character colour and the background colour. The background colour must be the same for all 1000 locations, but the foreground colour can change from one character cell to another.

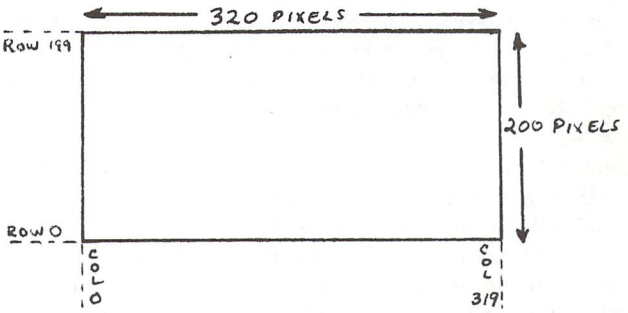
To help locate particular cells, it is convenient to number the rows from 0 to 24 and the columns from 0 to 39 as in the figure below.



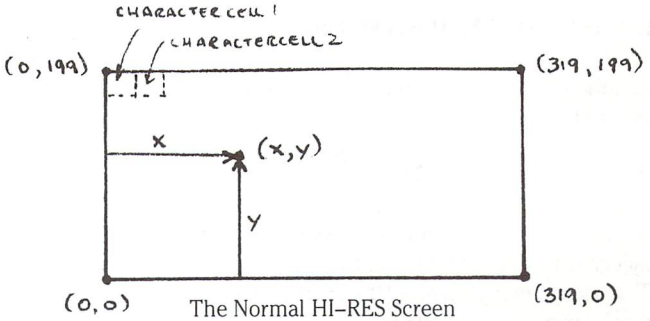
The text screen is fine for displaying character graphics, especially considering the 64's ability to place the character definitions in RAM where you can change the shape of the characters to virtually anything that you want. However for fine detailed graphics that involve the drawing of lines, circles, and other mathematical curves, there is a better solution – the HIRES screen.

The hires screen is made up of 200 rows, each containing 320 dots or 'pixels'. Each of these 64000 (i.e. 200*320) pixels can be turned 'on' or 'off' individually, allowing you to create very fine detailed pictures of enormous complexity.

To help locate specific pixels, it is convenient to number the rows from 0 to 199 and the columns from 0 to 319 as in the figure below.



I've chosen this particular numbering scheme because then we can think of the screen, in mathematical terms, as a Cartesian system, with the origin in the lower left hand corner. Any point on the screen can then be located by its (x,y) or Cartesian coordinates.

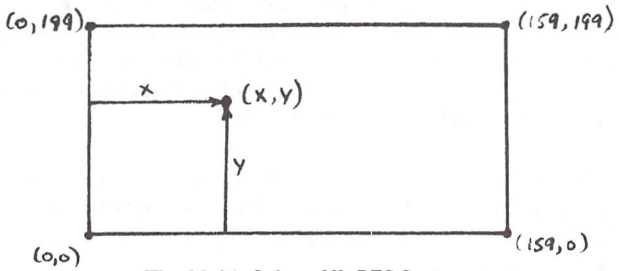


The Normal HI-RES Screen

For colour purposes, the hires screen is also arranged in a way similar to the normal text screen; that is, it can be thought of as containing 25 rows and 40 columns of 'character cells'. Each of these character cells is 8 pixels wide and 8 pixels deep, just like a regular character. Also, like the normal text screen, a character cell on the hires screen can contain only two colours. The 'off' pixels can be one colour, and the 'on' pixels another – these colours can of course be any of the 16 colours available on the text screen. Unlike the text screen, the background colour (i.e. the colour of the 'off' pixels) can change from one character cell to another as can the foreground colour.

This limitation of two colours can be overcome at the expense of a loss in resolution by switching to multi-colour mode. In this mode, 4 colours are possible within each character cell. Each coloured dot or pixel, however, is twice as wide as in normal hires mode. Thus the number of pixels across is 160. The vertical resolution is still 200. Also in this mode, the background colour must be the same for all 1000 character cells.

In multi-colour mode, more colourful pictures can be created, but the loss in resolution causes lines to be more jagged and curves to be less smooth. It will be up to you to determine which mode is more suitable for your application.



The Multi-Colour HI-RES Screen

As I mentioned earlier, all the graphic capabilities are available in BASIC through a series of complex PEEKS and POKES. Aside from the complexity involved, the time required to execute a command can be unbearable (eg. 30 seconds to clear the hires screen). Thus to be at all useful, the commands must be carried out in machine language with hopefully a simple interface to BASIC.

Listing 1. is a BASIC loader that, when run, will create a PRG file called HIRES on your disk. This is the machine language program that will contain all the graphic commands. Type this program in carefully, save it, and run it. The loader program contains a 'check' every 10 lines in an attempt to catch typing errors, but be careful, it is not foolproof, and a single typing error could make your program bomb. Listings 2, 3, and 4 are sample BASIC programs that make use of the commands. Before we look at these, let's go through the actual commands, their syntax, and the various options available.

1. Turning On The Hires Screen

To turn on the hires screen, a line with the following syntax is required.

```
100 SYS HIRES,0,BG,C1
```

In this line, BG is a number between 0 and 15 representing the background colour of the screen (note: 0 = black, 1 = white, etc.) and C1 is a number representing the plotting colour. Variables or numbers may be used for either BG or C1.

When this command is executed, the screen is cleared to the background colour and any points that are subsequently plotted or lines that are drawn (see below) will appear in the colour determined by C1.

It is also possible to turn on the hires screen without clearing it. The following line will do this.

```
100 SYS HIRES,0
```

(i.e. leave off the background and plotting colour)

This would be useful if your program needs to flip between the hires screen and the text screen and still retain any graphics on the hires screen. The first time you access hires graphics, you will, of course, want to clear the screen as well.

If you want the multi-colour screen, use the following line

```
100 SYS HIRES,1,BG,C1,C2,C3
```

where again BG is the background colour and C1, C2, and C3 are the three possible plotting colours (Remember, in multi-colour mode, four colours are possible in each character cell – the background colour, and three plotting colours). For those interested in technical details, plotting colour C1 corresponds to points plotted using the bitpair 01, C2 corresponds to the bitpair 10, and C3 corresponds to the bitpair 11.

Turning on the multi-colour screen without clearing it is done by

```
100 SYS HIRES,1
```

The graphics screen can be cleared at any time after the screen has been enabled by

```
100 SYS CLSCR,BG,C1                    (in hires mode)
```

and

```
100 SYS CLSCR,BG,C1,C2,C3              (in multi-colour mode)
```

2. Setting Plotting Colours

The background and plotting colours are set when hires mode is first turned on. They may, however, be changed later on by the command

```
200 SYS COLOUR,C1                      (in hires mode)
```

and

```
200 SYS COLOUR,C1,C2,C3                (in multi-colour mode)
```

In multi-colour mode, all three plotting colours must be included even if you only want to change C1.

3. Selecting A Plotting Colour

When graphics mode is enabled, colour C1 is the default plotting colour. In multi-colour mode, you may wish to change to colour C2, or colour C3, or back to colour C1 again. This is done by the command

```
300 SYS SELPC,C
```

where C = 1 if you want colour C1
C = 2 if you want colour C2
and C = 3 if you want colour C3

If executed when in normal hires mode, this command is ignored since there is only one plotting colour C1.

4. Setting The Drawing Mode

Any plotting or drawing in hires mode can be done in any of three ways.

ERASE mode – In this mode, the points and lines are erased rather than drawn (i.e. the pixels are turned off)

DRAW mode – In this mode, all points and lines are drawn (i.e. the pixels are turned on)

FLIP mode – In this mode, the condition of all points and lines are flipped (i.e. the pixels that are on are turned off and vice–versa)

To set a particular drawing mode, simply include

```
400 SYS DMODE,M
```

where M = 0 if you want erase mode
M = 1 if you want draw mode
and M = 2 if you want flip mode

5. Plotting A Point

To plot a point at coordinates (X,Y) (Remember: (0,0) is in the lower left–hand corner of the screen), use

```
500 SYS PLOT,X,Y
```

The point will be plotted in the current plotting colour (as selected in 3. above) in the current drawing mode (as set by 4. above). If you wish, you can include up to two additional parameters.

```
500 SYS PLOT,X,Y,C
```

will select plotting colour C before plotting the point at (X,Y) and

```
500 SYS PLOT,X,Y,C,M
```

will select plotting colour C and drawing mode M before plotting at (X,Y).

6. Moving The Drawing Cursor

Whenever a point is plotted or a line is drawn, the drawing routines remembers the coordinates of the last point plotted. This point is called the drawing cursor, and is used in the DRAW TO command described below. It is possible to move the drawing cursor to a new position with coordinates (X,Y) and without plotting or drawing by the command

```
600 SYS MOVE,X,Y
```

7. Drawing Lines

This is the most versatile of the commands and offers the most options.

(i) The DRAW command

```
700 SYS DRAW,X,Y
```

will draw a straight line from the current position of the drawing cursor (i.e. the last point plotted) to the point with coordinates (X,Y). It will also set the new position of the drawing cursor to (X,Y). The line will be drawn in the current plotting colour and in the current drawing mode. As with the PLOT command, the plotting colour and the drawing mode may be changed before the line is drawn by the addition of one or two more parameters; that is,

```
700 SYS DRAW,X,Y,C
```

will draw the line to (X,Y) in plotting colour C, while

```
700 SYS DRAW,X,Y,C,M
```

will draw the line in plotting colour C and in drawing mode M.

(ii) The DRAW . . . TO . . . command

```
700 SYS DRAW,X1,Y1 TO X2,Y2
```

will draw a line from the point (X1,Y1) to the point (X2,Y2) and set the drawing cursor to (X2,Y2) (The initial position of the drawing cursor is ignored). As above, you can have

```
700 SYS DRAW,X1,Y1 TO X2,Y2,C
```

or

```
700 SYS DRAW,X1,Y1 TO X2,Y2,C,M
```

You can even have

```
700 SYS DRAW,X1,Y1,C,M TO X2,Y2
```

or

```
700 SYS DRAW,X1,Y1,C,M TO X2,Y2,D,N
```

In the latter case, the C,M will, in fact, be ignored and the line will be drawn in plotting colour D and in drawing mode N.

(iii) The DRAW . . . TO . . . TO . . . TO . . . command

This feature makes this command similar to the HPLOT command on the APPLE. and allows you to draw a straight line from one point to another, then from that point to another, and so on. The syntax is . . .

```
700 SYS DRAW,X1,Y1 TO X2,Y2 TO X3,Y3 TO X4,Y4
```

with as many points Xi,Yi as you can fit into a BASIC line. This command is useful for drawing shapes that can be made up of straight lines, eg. parallelograms, hexagons, etc. As above you can include ,C,M anywhere after an Xi,Yi but keep in mind that it won't take effect until the line is drawn to that particular point.

8. Drawing Rectangles

A rectangle may be drawn using the command

```
800 SYS BOX,X,Y,WIDTH,HEIGHT
```

where (X,Y) are the coordinates of the top left–hand corner of the rectangle, and WIDTH and HEIGHT are the width and height of the rectangle respectively. As usual, you may optionally have

```
800 SYS BOX,X,Y,WIDTH,HEIGHT,C
```

or

```
800 SYS BOX,X,Y,WIDTH,HEIGHT,C,M
```

9. Restoring The Text Screen

If you wish to return to normal text mode from graphics mode, you can do so by

```
900 SYS TEXT
```

This will return you to the text screen with exactly the same conditions that prevailed prior to entering graphics mode (eg. if you were in upper/lower case prior to entering graphics mode, you will still be there when you return to text mode). Used in conjunction with the command SYS HIRES,0 or SYS HIRES,1 this can be used for flipping between the graphics screen and the text screen.

10. Printing To The Graphics Screens

Text or character graphics can be printed to the hires screen using the PRNT command. For example,

```
1000 SYS PRNT,C,R,A$
```

will print whatever is in A$ onto the hires screen, starting in column C of row R.

Notes: 1. Printing takes place only in the 'character cells'. Therefore R and C are the row and column numbers as determined by the text screen, not the Cartesian coordinates on the hires screen (i.e. 0 <= R <= 24 and 0 <= C <= 39 with (0,0) in the top left-hand corner of the screen).

2. The string to be printed may be a string variable, a string of characters between quotes, or even a string expression using MID$, LEFT$, or RIGHT$. The string expression may involve concatenation, for example:

```
1000 SYS PRNT,5,10,A$ + " DONE "
```

but don't forget the '+' sign

```
1000 SYS PRNT,5,10,A$; " DONE "
```

will yield an error.

3. The characters to be printed can be anything that you would normally use in a PRINT statement on the text screen, including cursor control characters such as cursor left, etc. , RVS ON/OFF, colour control characters such as CTRL-1 (for white), etc. You can even use CNTRL-N to switch to lowercase and CHR$(142) to switch to uppercase. Only CLR and HOME for clearing the screen and homing the cursor are ignored.

Cursor up and down work properly, but be careful when trying to cursor up beyond the top line of the screen or down below the bottom line. Strange things will result, but the program will not crash.

4. The text can be printed in either hires mode or multi-colour mode. In multi-colour mode, the characters will look a little funny and may be unreadable depending on the settings for the three plotting colours. If you set C1 and C3 to the colour you want the characters to be and C2 to the background colour, the characters are perfectly readable. By changing C2 and C3, you can get some interesting effects (see the sample programs for an illustration). The best solution is to create your own custom characters using a multi-colour character editor and use those characters instead (see below).

11. Changing Character Sets

Text can be printed in upper case/graphics, upper/lower case, and even a combination of the two (something that can't be done on the normal text screen without using raster interrupts). It is even possible to print characters that you created with a character editor and stored in RAM. To print such characters, simply precede the printing with the command

```
1100 SYS CHSET,AD
```

where AD is the address of the RAM character set. Because of the way character sets work, AD must be a multiple of 2048. Thus

```
1100 SYS CHSET,7*2048
```

would be required if your character set were stored in memory at address 14336 (=7*2048).

Note also that AD=0 will choose the ROM character set containing upper case and graphic characters while AD=1 will choose the upper/lower case characters in ROM.

12. The TRAP Command

When drawing mathematical curves, errors such as 'division by zero', 'illegal quantity', etc. can quickly halt a program. When this happens, the TRAP command, which is the equivalent of the ONERRGOTO command on the APPLE, can overcome this. For example,

```
1200 SYS TRAP,1500
```

once executed, will cause control to be transferred to line 1500 whenever an error occurs. The routine at line 1500 could then check for the type of error and take appropriate action. The important point is that the program will not stop unless you tell it to.

To check for the type of error, include the following in your error handling routine.

```
1500 X=PEEK(781)
```

This will be the error number that would normally be passed on the BASIC interpreter.

eg. if X=20 then a division by zero error occurred
if X=14 then an illegal quantity error occurred
etc.

The TRAP command can be disabled (i.e. the BASIC interpreter handles all errors) by leaving off the line number in the TRAP command, ie.

```
1300 SYS TRAP
```

Caution: When an error occurs, the stack pointer can be in an unpredictable position. For this reason, before sending control to the line of your error handling routine, the stack is cleared. This includes all information about FOR . . . NEXT loops and RETURN addresses of subroutines. Thus after handling a specific error, you must not go into the middle of a FOR . . . NEXT loop or the middle of a subroutine, even if that is where the error originally occurred.

Listing 2 is a short demonstration program that shows how easy it is to make a pie chart. Notice how the HIRES routines are loaded in line 20, and the variables, that are the addresses of the various commands, are initialized in lines 120 through 150.

At this time, there is no command for drawing circles in my routines. When it is complete, there will be such a command with the following syntax.

SYS CIRCLE,XC,YC,XR,YR,SA,EA,INC

where XC,YC are the coordinates of the centre of the circle
XR is the radius in the horizontal direction
YR is the radius in the vertical direction
SA is the starting angle in degrees (i.e. the angle with the horizontal where the plotting will start)
EA is the ending angle in degrees (i.e. the angle where the plotting will end)
and INC is the increment in degrees

Note: The circle is drawn as a series of straight lines (and so is really a polygon). INC determines how many sides this polygon has, and hence how 'smooth' the circle is.

Since the command is not implemented yet, a BASIC equivalent is given in line 500 of Listing 2. To draw a circle then, it is only necessary to initialize the variables XC,YC,XR,YR,SA,EA, and INC and then GOSUB500.

Notice that to get a complete circle, the starting angle and the ending angle must be 360 degrees apart. Also by making INC equal to 120 degrees, 60 degrees, and 72 degrees, you can draw a triangle, hexagon, and pentagon respectively.

Listing 3 is another demo that draws a bar chart, and illustrates what printing looks like in multi-colour mode. Experiment with lines 460, 480 and 520 and observe the effects.

Listing 4 is a program that draws the graph of a mathematical curve. When typing it in, put a REM in front of each TRAP command. This will allow you to debug the program more easily – otherwise the TRAP command will trap all your typing errors and it can become frustrating.

The program draws the sinusoidal curve

$$y = \sin(2*x) + \cos(3*x) \text{ (line 220)}$$

After the program is debugged, try drawing the curves

$$y = 1/\sin(x)$$

or

$$y = \sin(x)/\cos(x)$$

or

$$y = \text{sqr}(4-x*x)$$

Try it with and without the REM statements in front of the TRAP commands.

This program can be described as a general purpose drawing routine; that is, it will draw the graph of virtually any function. As such, it is somewhat slow. The plot can be speeded up by decreasing the number of plotting points (line 770) but this will also have a negative effect in that the curve will be more jagged. Special purpose routines, for drawing graphs that do not have undefined regions and for which all plotting occurs within the specified range will proceed much quicker.

One Final Note On Memory Usage

The hires screen is located at $E000 (i.e. underneath the kernal ROM). As such it does not steal any memory from BASIC. The routines themselves are stored in the $C000 block from $C000 to $C81E. Colour memory for the hires screen is located at $CC00 and extends to $CFFF. Since the DOS wedge also uses this area, these routines will not work with the wedge installed. Similarly, it may not work with other utilities that use the $C000 block of memory.

Listing 1

```
BC 100 open 1,8,15,"i0":close1
HF 110 open 1,8,1,"@0:hires"
LG 120 print#1,chr$(0);:print#1,chr$(192);
BJ 130 for j=1 to 2080
DG 140 read x : print#1,chr$(x);
GL 150 chk=chk+x
CI 160 next j
GO 170 if chk<>245727 then print "has an error":
       close1:end
HI 180 close1
PL 190 print "save successful":end
JI 1000 data 76,194,193, 76,247,195, 76, 98
BG 1010 data 195, 76,110,194, 76, 30,194, 76
PO 1020 data 214,196, 76,228,196, 76, 11,197
AG 1030 data 76, 67,197, 76,169,192, 76,206
JL 1040 data 197, 76,199,199, 76, 4,200, 0
IO 1050 data 0, 0, 0, 0, 0, 0, 0, 0
JC 1060 data 0, 0, 0,255,128, 0, 7,248
MP 1070 data 0, 0, 0, 0, 0, 0, 0, 0
CH 1080 data 0, 1, 0, 15,240,240, 0, 0
ID 1090 data 208, 0, 0, 0, 0,173, 58,192
DI 1100 data 208, 27,173, 0,221,141, 57,192
IA 1110 data 173, 24,208,141, 58,192,173, 17
HK 1120 data 208,141, 59,192,173, 22,208,141
FD 1130 data 60,192, 32,110,192, 96,173, 0
MF 1140 data 3,201,231,208, 7,173, 1, 3
HI 1150 data 201,192,240, 44,173, 0, 3,141
JM 1160 data 234,192,173, 1, 3,141,235,192
HM 1170 data 169,231,141, 0, 3,169,192,141
MI 1180 data 1, 3,173, 2, 3,141, 41,193
PH 1190 data 173, 3, 3,141, 42,193,169, 8
JN 1200 data 141, 2, 3,169,193,141, 3, 3
PO 1210 data 96,173, 58,192,240, 26,141, 24
FG 1220 data 208,173, 57,192,141, 0,221,173
GH 1230 data 59,192,141, 17,208,173, 60,192
HD 1240 data 141, 22,208,169, 0,141, 58,192
GG 1250 data 96, 72,169,127,141, 13,220,165
GO 1260 data 1,141, 56,192, 41,253,133, 1
FN 1270 data 104, 96, 72,173, 56,192,133, 1
KJ 1280 data 169,129,141, 13,220,104, 96, 16
NG 1290 data 3, 76,139,227,142, 13, 3, 44
BC 1300 data 76,192, 16,245,169, 0,133, 20
MG 1310 data 169, 0,133, 21,162,250,154,169
DP 1320 data 167, 72,169,233, 72, 76,163,168
BC 1330 data 32,169,192,173,234,192,141, 0
JC 1340 data 3,173,235,192,141, 1, 3,173
EJ 1350 data 41,193,141, 2, 3,173, 42,193
GL 1360 data 141, 3, 3,169, 0,141, 76,192
HL 1370 data 76,131,164,164,254,240, 13,160
GB 1380 data 0,145,251,200,208,251,230,252
FH 1390 data 198,254,208,243,164,253,240, 10
BH 1400 data 136,240, 5,145,251,136,208,251
BJ 1410 data 145,251, 96, 32,201,192,160, 0
HK 1420 data 132,251,160,204,132,252,160,232
JJ 1430 data 132,253,160, 3,132,254, 32, 43
NK 1440 data 193,169, 0,133,251,169,224,133
JB 1450 data 252,169, 64,133,253,169, 31,133
EO 1460 data 254,169, 0, 32, 43,193, 76,218
```

```
KG 1470 data 192,  32, 253, 174,  32, 138, 173,  32
PA 1480 data 247, 183, 166,  21, 165,  20,  96,  32
GC 1490 data 253, 174,  32, 124, 193, 141,  43, 192
CF 1500 data 142,  44, 192,  32, 121, 193, 141,  45
IF 1510 data 192, 142,  46, 192, 169,  63, 162,   1
ID 1520 data  44,  53, 192,  16,   4, 169, 159, 162
BA 1530 data   0, 205,  43, 192, 138, 237,  44, 192
LG 1540 data 176,   3,  76,  72, 178, 169, 199, 205
MI 1550 data  45, 192, 169,   0, 237,  46, 192, 144
KF 1560 data 241,  96,  32,  77, 192,  32, 121, 193
PB 1570 data 240,   2, 169, 128, 141,  53, 192,  32
FB 1580 data 121,   0, 240,   3,  32,  30, 194, 173
GJ 1590 data   0, 221,   9,   3,  73,   3, 141,   0
GI 1600 data 221, 173,  24, 208,  41,   7,   9,   8
EF 1610 data   9,  48, 141,  24, 208, 173,  17, 208
LG 1620 data   9,  32, 141,  17, 208,  44,  53, 192
BI 1630 data  16,  12, 173,  22, 208,   9,  16, 141
GI 1640 data  22, 208, 169,   3, 208,  10, 173,  22
IK 1650 data 208,  41, 239, 141,  22, 208, 169,   7
MA 1660 data 141,  54, 192,  73, 255, 141,  55, 192
IO 1670 data 169, 255, 141,  51, 192,  96, 169,   1
ME 1680 data 141,  65, 192, 173,  67, 192, 141,  66
KO 1690 data 192, 169, 128, 141,  52, 192,  32, 135
JJ 1700 data 193, 173,  45, 192,  10,  10,  10,  10
OD 1710 data 141,  62, 192, 141,  70, 192, 173,  43
AK 1720 data 192,  41,  15, 141,  61, 192,  44,  53
NO 1730 data 192,  48,  12,  13,  62, 192, 141,  62
HH 1740 data 192, 141,  70, 192,  76,  75, 193, 141
MN 1750 data  33, 208,  32, 121, 193,  41,  15, 141
AF 1760 data  63, 192,  32, 121, 193, 141,  64, 192
HD 1770 data 173,  62, 192,  76,  75, 193,  32, 135
JC 1780 data 193, 162,   3, 189,  43, 192, 157,  39
MM 1790 data 192, 202,  16, 247,  96,  56, 169, 199
DD 1800 data 237,  41, 192,  72,  74,  74,  74, 133
JF 1810 data 252, 160,   0, 132, 251,  74, 102, 251
CD 1820 data  74, 102, 251, 101, 252, 133, 252, 173
GJ 1830 data  39, 192, 174,  40, 192,  45,  55, 192
HL 1840 data  44,  53, 192,  16,   6,  10,  72, 138
DC 1850 data  42, 170, 104,  24, 101, 251, 133, 251
AM 1860 data 138, 101, 252, 133, 252, 104,  41,   7
NF 1870 data  24, 101, 251, 133, 251, 133, 253, 144
OI 1880 data   2, 230, 252, 165, 252,  74, 102, 253
IH 1890 data  74, 102, 253,  74, 102, 253, 133, 254
GC 1900 data  44,  53, 192,  48,  16,  24, 169,   0
EI 1910 data 101, 253, 133, 253, 169, 204, 101, 254
LB 1920 data 133, 254,  76, 249, 194, 173,  65, 192
BD 1930 data 201,   3, 144, 234,  24, 169,   0, 101
GL 1940 data 253, 133, 253, 169, 216, 101, 254, 133
JD 1950 data 254,  24, 165, 251, 105,   0, 133, 251
JL 1960 data 165, 252, 105, 224, 133, 252, 173,  39
MK 1970 data 192,  45,  54, 192, 170,  96, 169,   0
IO 1980 data 168,  44,  52, 192,  16,   4, 112,  20
EB 1990 data  80,  15,  36,   2,  48,   9, 169, 255
EE 2000 data 133,   2,  36, 107,  48,   1,  96, 177
CB 2010 data 251,  77,  51, 192,  44,  53, 192,  48
DA 2020 data  10,  61,  86, 195, 133,  97, 189,  86
HD 2030 data 195, 208,   8,  61,  94, 195, 133,  97
JD 2040 data 189,  94, 195,  73, 255,  49, 251,   5
EP 2050 data  97, 145, 251, 177, 253,  45,  66, 192
AC 2060 data  13,  70, 192, 145, 253,  96, 128,  64
DJ 2070 data  32,  16,   8,   4,   2,   1, 192,  48
EK 2080 data  12,   3,  32, 110, 194,  32, 121,   0
HO 2090 data 240,  11,  32, 228, 196,  32, 121,   0
MD 2100 data 240,   3,  32, 214, 196,  32, 201, 192
CI 2110 data  32, 125, 194,  32,  14, 195,  76, 218
BI 2120 data 192, 169,   1, 149, 106, 169,   0, 149
AA 2130 data 107,  56, 189,  43, 192, 253,  39, 192
CC 2140 data 149,  98, 189,  44, 192, 253,  40, 192
OA 2150 data 149,  99,  16,  20, 169, 255, 149, 106
BO 2160 data 149, 107,  56, 169,   0, 245,  98, 149
BF 2170 data  98, 169,   0, 245,  99, 149,  99,  96
DK 2180 data  21,  98, 208,   4, 149, 106, 149, 107
KP 2190 data  96, 165,  99,  74, 133, 103, 165,  98
MN 2200 data 106, 133, 102,  24, 169,   0, 229,  98
MA 2210 data 133, 104, 169,   0, 229,  99, 133, 105
AI 2220 data  96,  24, 165, 102, 101, 100, 133, 102
HL 2230 data 170, 165, 103, 101, 101, 133, 103, 197
BC 2240 data  99, 144,  19, 208,   4, 228,  98, 144
NF 2250 data  13, 138,  56, 229,  98, 133, 102, 165
CC 2260 data 103, 229,  99, 133, 103,  56,  96,  32
OO 2270 data 135, 193,  32, 121,   0, 240,  44, 201
DF 2280 data 164, 208,  16,  32, 113, 194,  32, 115
EM 2290 data   0,  32, 138, 193,  32, 121,   0, 201
NB 2300 data  44, 208,  13,  32, 228, 196,  32, 121
GP 2310 data   0, 201,  44, 208,   3,  32, 214, 196
EO 2320 data  32,  43, 196,  32, 121,   0, 201, 164
AC 2330 data 240, 220,  96,  32, 201, 192, 162,   0
PB 2340 data 134,   2,  32, 129, 195, 162,   2,  32
HO 2350 data 129, 195, 165,  98, 197, 100, 165,  99
LL 2360 data 229, 101, 144,  62,  32, 185, 195,  36
ME 2370 data 107,  16,  10,  32, 113, 194,  56, 169
JJ 2380 data   0, 229, 108, 133, 108,  32, 125, 194
GE 2390 data  32,  14, 195, 230, 104, 208,   4, 230
DB 2400 data 105, 240, 102, 238,  39, 192, 208,   3
EJ 2410 data 238,  40, 192,  32, 209, 195, 144,   9
OM 2420 data  24, 173,  41, 192, 101, 108, 141,  41
JN 2430 data 192,  32, 125, 194,  32,  14, 195,  76
MO 2440 data  91, 196, 162,   1, 181,  98, 180, 100
NC 2450 data 149, 100, 148,  98, 202,  16, 245,  32
GO 2460 data 185, 195,  36, 107,  16,  10,  32, 113
HG 2470 data 194,  56, 169,   0, 229, 108, 133, 108
DD 2480 data  32, 125, 194,  32,  14, 195, 230, 104
MA 2490 data 240,  31,  24, 173,  41, 192, 101, 108
LO 2500 data 141,  41, 192,  32, 209, 195, 144,   8
KE 2510 data 238,  39, 192, 208,   3, 238,  40, 192
LC 2520 data  32, 125, 194,  32,  14, 195,  76, 166
AE 2530 data 196,  36, 107,  16,   3,  32,  14, 195
NH 2540 data  32, 113, 194,  76, 218, 192,  32, 121
NB 2550 data 193,  41,   3,  73,   3, 106, 106, 106
JC 2560 data 141,  52, 192,  96,  32, 121, 193,  41
FJ 2570 data   3, 240,  27,  44,  53, 192,  16,  22
KH 2580 data 141,  65, 192, 170, 189,  61, 192, 141
KN 2590 data  70, 192, 189,  66, 192, 141,  66, 192
NF 2600 data 189,   7, 197, 141,  51, 192,  96,   0
PE 2610 data  85, 170, 255,  32, 121, 193,  10,  10
KA 2620 data  10,  10, 141,  62, 192,  44,  53, 192
FK 2630 data  48,   9,  13,  61, 192, 141,  62, 192
CE 2640 data  76,  51, 197,  32, 121, 193,  41,  15
KG 2650 data 141,  63, 192,  32, 121, 193,  41,  15
AD 2660 data 141,  64, 192, 174,  65, 192, 189,  61
AO 2670 data 192, 141,  70, 192, 189,  66, 192, 141
AL 2680 data  66, 192,  96,  32, 110, 194,  32, 135
JL 2690 data 193, 162,   3, 189,  43, 192, 157,  47
EL 2700 data 192, 202,  16, 247,  32, 121,   0, 240
BF 2710 data  11,  32, 228, 196,  32, 121,   0, 240
IK 2720 data   3,  32, 214, 196,  24, 173,  39, 192
ND 2730 data 109,  47, 192, 141,  43, 192, 173,  40
LA 2740 data 192, 109,  48, 192, 141,  44, 192, 173
EE 2750 data  41, 192, 141,  45, 192, 173,  42, 192
OP 2760 data 141,  46, 192,  32, 156, 193,  32,  43
HJ 2770 data 196,  56, 173,  45, 192, 237,  49, 192
PI 2780 data 141,  45, 192, 173,  46, 192, 237,  50
NI 2790 data 192, 141,  46, 192,  32, 181, 193,  32
GD 2800 data  43, 196,  56, 173,  43, 192, 237,  47
LF 2810 data 192, 141,  43, 192, 173,  44, 192, 237
BG 2820 data  48, 192, 141,  44, 192,  32,  43, 196
KJ 2830 data  24, 173,  45, 192, 109,  49, 192, 141
DM 2840 data  45, 192, 173,  46, 192, 109,  50, 192
AD 2850 data 141,  46, 192,  76,  43, 196, 169,   0
GD 2860 data 133, 251, 133, 252,  32, 241, 183, 224
```

```
HL 2870 data  40,144,   3,  76,  72,178,142,  73
KJ 2880 data 192,  32,241,183,142,  74,192,138
GO 2890 data 240,  18,224,  25,176,237,  24,165
IN 2900 data 251, 105,  40,133,251,144,   2,230
PI 2910 data 252, 202,208,242,  24,173,  73,192
MH 2920 data 101, 251,133,251,133,253,133,   3
IF 2930 data 169,   0,101,252,133,252,  24,  72
NF 2940 data 105, 216,133,254,104,105,204,133
MM 2950 data   4,   6,251,  38,252,   6,251,  38
PK 2960 data 252,   6,251,  38,252,  24,165,252
PM 2970 data 105, 224,133,252,  32,253,174,  32
PO 2980 data 158, 173,  32,143,173,  32,166,182
KI 2990 data 170, 160,   0,232,202,208,   1,  96
HL 3000 data 177,  34,  32,  73,198,200,  76,  60
NA 3010 data 198, 133,215,138,  72,152,  72,165
LM 3020 data 215,  48,  17,201,  32,144,  28,201
FH 3030 data  96, 144,   4,  41,223,208,   2,  41
FF 3040 data  63,  76,110,199,  41,127,201,127
MO 3050 data 208,   2,169,  94,201,  32,144,125
JC 3060 data  76, 108,199,201,  14,208,   6,  32
CI 3070 data 219, 199,  76,160,199,201,  17,208
HC 3080 data  11, 162,  40,  32,  71,199,202,208
AI 3090 data 250,  76,160,199,201,  18,208,   8
EF 3100 data 169,   1,141,  75,192,  76,160,199
JL 3110 data 201,  29,208,   6,  32,  71,199,  76
BF 3120 data 160, 199,162,   3,  44,162,  15,221
FG 3130 data 205, 198,240,   6,202,  16,248,  76
JO 3140 data 160, 199,189,221,198,  10,  10,  10
IF 3150 data  10, 141,  62,192,  44,  53,192,  48
MK 3160 data   6,  13,  61,192,141,  62,192,  32
MM 3170 data  51, 197,  76,160,199,   5,  28,  30
HE 3180 data  31,  16,  28,  30,  31,   1,  21,  22
CB 3190 data  23,  24,  25,  26,  27,   1,   2,   5
AA 3200 data   6,   0,   4,   7,   3,   8,   9,  10
FI 3210 data  11,  12,  13,  14,  15,201,  14,208
PM 3220 data   6,  32,216,199,  76,160,199,201
IM 3230 data  17, 208,  11,162,  40,  32,  28,199
KA 3240 data 202, 208,250,  76,160,199,201,  18
IB 3250 data 208,   8,169,   0,141,  75,192,  76
GI 3260 data 160, 199,201,  29,208,143,  32,  28
NO 3270 data 199,  76,160,199,165,253,208,   2
MC 3280 data 198, 254,198,253,165,   3,208,   2
MB 3290 data 198,   4,198,   3,  56,165,251,233
KM 3300 data   8, 133,251,165,252,233,   0,133
BP 3310 data 252, 165,251,201,   0,165,252,233
CC 3320 data 224, 176,   3,  32,  71,199,  96,230
KN 3330 data 253, 208,   2,230,254,230,   3,208
PI 3340 data   2, 230,   4,  24,169,   8,101,251
DA 3350 data 133, 251,144,   2,230,252,165,251
GP 3360 data 201,  64,165,252,233,255,144,   3
EP 3370 data  32,  28,199,  96,   9,  64,174,  75
NG 3380 data 192, 240,   2,   9,128,  32,165,199
ID 3390 data 160,   7,  32,230,199,177,   5,145
KI 3400 data 251, 136,  16,249,  32,245,199,200
HJ 3410 data 173,  61,192,  44,  53,192,  16,   8
BO 3420 data 173,  64,192,145,253,173,  63,192
BJ 3430 data  13,  62,192,145,   3,  32,  71,199
CC 3440 data 104, 168,104,170,  96,133,   5,169
KH 3450 data   0, 133,   6,   6,   5,  38,   6,   6
GK 3460 data   5,  38,   6,   6,   5,  38,   6,  24
IK 3470 data 173,  71,192,101,   5,133,   5,173
KE 3480 data  72, 192,101,   6,133,   6,  96,  32
AA 3490 data 253, 174,  32,138,173,  32,247,183
ON 3500 data 166,  21,208,   9,165,  20,208,   3
BA 3510 data 162, 208,  44,162,216,142,  72,192
LK 3520 data 162,   0,142,  71,192,  96,173,  14
AN 3530 data 220,  41,254,141,  14,220,165,   1
IB 3540 data  41, 251,133,   1,  96,165,   1,   9
CM 3550 data   4, 133,   1,173,  14,220,   9,   1
GL 3560 data 141,  14,220,  96,  32,121,   0,240
LC 3570 data  15,  32,110,192,  32,121,193,141
OC 3580 data 245, 192,142,249,192,169,128,  44
CB 3590 data 169,   0,141,  76,192,  96,   0,   0
```

Listing 2

```
JO  10 print" S ":poke 53280,5:poke 53281,1
LF  20 if peek(49152)<>76 then load "hires",8,1
CJ  30 :
HG  100 rem initialize variables
CO  110 :
EF  120 hires = 12*4096:draw = hi + 3:plot = dr + 3
CJ  130 move = pl + 3:clscr = mo + 3:dmode = cl + 3
AC  140 selpc = dm + 3:colour = se + 3:box = co + 3
DL  150 text = bo + 3:prnt = te + 3:chset = pr + 3:trap = ch + 3
EB  160 :
NJ  170 rem begin the show
IC  180 :
MH  190 sys hires,0,1,6
IJ  200 xc = 159:yc = 100:xr = 70:yr = 50:inc = 10
PG  210 sa = 45:ea = 75:gosub 620
FD  220 sa = 75:ea = 160:gosub 620
BH  230 sa = 160:ea = 240:gosub 620
EI  240 sa = 240:ea = 325:gosub 620
GA  250 sys colour,7
EG  260 xc = 175:sa = -35:ea = 45:gosub 620
OB  270 sys colour,9
LF  280 sys box,6,170,307,165
EG  290 sys box,3,172,313,169
LA  300 sys chset,1
KP  310 a$ = "r" + chr$(30) + "Rent"    :rem rvs + grn
DG  320 sys prnt,15,9,a$
JG  330 a$ = chr$(156) + "Food"      :rem pur
KA  340 sys prnt,13,13,a$
BO  350 a$ = chr$(28) + "Clothes"    :rem red
PC  360 sys prnt,18,16,a$
LB  370 a$ = chr$(158) + "Travel" + chr$(142)  :rem yel +
    upper/graphics
DD  380 sys prnt,24,12,a$
HC  390 a$ = chr$(154) + chr$(176) + "CCCCCCCCCCCCCC
        CCCCCCCCCCCCCCCCCCCCCCC" + chr$(174)
GH  400 sys prnt,0,0,a$
IM  410 a$ = chr$(221) + "[9 spaces]nP PIE CHARTS are "
        + chr$(28) + "Easy Z [10 spaces]N "
CK  420 sys prnt,0,1,a$ + chr$(221)
LE  430 a$ = chr$(173) + "CCCCCCCCCCCCCCCCCCCCC
        CCCCCCCCCCCCCCCC" + chr$(189) + " R "
CK  440 sys prnt,0,2,a$
AO  450 get a$:if a$<>chr$(13) then 450
MM  460 end
KE  470 :
PE  480 rem draw arc
OF  490 :
CI  500 z1 = sa*π/180:z2 = ea*π/180:z3 = inc*π/180
NI  510 x = xc + xr*cos(z1):y = yc + yr*sin(z1)
JA  520 sys move,x,y
FC  530 for i = z1 to z2 step z3
AM  540 x = xc + xr*cos(i):y = yc + yr*sin(i)
HC  550 sys draw,x,y
ED  560 next
AD  570 sys draw,xc + xr*cos(z2),yc + yr*sin(z2)
AG  580 return
CM  590 :
CN  600 rem draw pie
GN  610 :
FB  620 gosub 500
LE  630 sys draw,xc,yc
EH  640 sys draw,xc + xr*cos(z1),yc + yr*sin(z1)
GK  650 return
```

Listing 3

```
JO  10 print" S ":poke 53280,5:poke 53281,1
LF  20 if peek(49152)<>76 then load "hires",8,1
CJ  30 :
HG  100 rem initialize variables
CO  110 :
EF  120 hires=12*4096:draw=hi+3:plot=dr+3
CJ  130 move=pl+3:clscr=mo+3:dmode=cl+3
AC  140 selpc=dm+3:colour=se+3:box=co+3
DL  150 text=bo+3:prnt=te+3:chset=pr+3:trap=ch+3
EB  160 :
NJ  170 rem begin the show
IC  180 :
JE  190 sys hires,1,0,1,2,6
MI  200 sys dmode,1:sys selpc,1
DJ  210 sys draw,33,168 to 33,87 to 133,87
PB  220 sys draw,33,86 to 133,86
JK  230 for k=1 to 5
NE  240 for j=37 to 133 step 4
GK  250 sys plot,j,84+16*k
BO  260 nextj,k
AN  270 for k=0 to 5
FN  280 y(k)=rnd(1)*80+88:c=rnd(1)*8+1
KE  290 sys colour,1,c,6
BP  300 for i=0 to 8
JJ  310 s=43+k*16+i
CG  320 sys draw,s,88 to s,y(k),2
JC  330 next i
JD  340 next k
BA  350 sys move,34,88
KC  360 for k=0 to 5
HL  370 s=47+k*16
HC  380 sys draw,s,y(k),3
LG  390 next k
CG  400 sys colour,1,0,1
PE  410 sys prnt,11,15,"79 80 81 82 83 84"
KA  420 a$="10 8 6 4 2"
PI  430 for i=1 to len(a$) step 2
JP  440 sys prnt,6,3+i,mid$(a$,i,2)
GM  450 next
CK  460 sys colour,2,0,2
ND  470 sys prnt,14,2,"annual sales"
KL  480 sys colour,0,0,6
CK  490 a$="" : for k=1 to 21 : a$=a$+chr$(164) : next
CD  500 sys prnt,9,17,a$
DO  510 sys prnt,9,18," r b a r  g r a p h s R "
KA  520 sys colour,5,7,7
AH  530 sys prnt,12,20,"a r e  n i c e"
BK  540 sys prnt,2,22,"i n  m u l t i - c o l o r  m o d e"
FE  550 get a$:if a$<>chr$(13) then 550
```

Listing 4

```
JO  10 print" S ":poke 53280,5:poke 53281,1
LF  20 if peek(49152)<>76 then load "hires",8,1
CJ  30 :
HG  100 rem initialize variables
CO  110 :
EF  120 hires=12*4096:draw=hi+3:plot=dr+3
CJ  130 move=pl+3:clscr=mo+3:dmode=cl+3
AC  140 selpc=dm+3:colour=se+3:box=co+3
DL  150 text=bo+3:prnt=te+3:chset=pr+3:trap=ch+3
EB  160 :
IK  170 rem plot the graph
IC  180 :
HA  190 x=x+dx:if x>xax then 650
EP  200 sys trap,400        : rem catch any calculation errors
DM  210 y=sin(2*x) + cos(3*x) : rem insert function to be
        graphed here
JI  220 sys trap,450        : rem now catch any plotting errors
OP  230 xp=(x-xin)*sx       : rem  x-coordinate for plot
DB  240 yp=(y-yin)*sy       : rem  y-coordinate for plot
PF  250 on pf goto 360,420
IH  260 :
JP  270 rem last point was plotted ok,
AJ  280 rem so draw from last point to current point
GJ  290 :
CE  300 sys draw,xp,yp
PO  310 goto 190
EL  320 :
KN  330 rem last point was out of range,
HP  340 rem so draw from boundary to current point
CN  350 :
JD  360 sys draw,xp,boundary to xp,yp
IE  370 pf=0:goto 190
AP  380 :
AL  390 rem last point was not defined,
KA  400 rem so just plot the current point
OA  410 :
OM  420 sys plot,xp,yp
EI  430 pf=0:goto 190
MC  440 :
IP  450 rem something went wrong with the function
AE  460 :
PN  470 err=peek(781):pf=2
CN  480 if err=14 then 190 : rem  illegal quantity error
MH  490 if err=20 then 190 : rem  division by zero error
ID  500 if err=16 then 190 : rem  overflow error
EP  510 if err=11 then sys prnt,3,23," syntax error
        in function definition" :goto 660
BF  520 sys prnt,4,23," oops! i forgot about error # "
        +str$(err):goto 660
GI  530 :
JD  540 rem  tried to plot out of range. x-coord. should be o.k.
NF  550 rem  therefore, just test the y-coord.
EK  560 :
LL  570 err=peek(781):if err<>14 then 520
JA  580 if yp>199 then boundary=199 : rem  point is above
        top of screen
GC  590 if yp<0 then boundary=0 : rem  point is below
        bottom of screen
HN  600 if pf=0 then sys draw,xp-dx,boundary
JD  610 pf=1:goto 190
AO  620 :
NE  630 rem  end by pressing <return>
EP  640 :
CH  650 sys prnt,13,23," graph completed "
GL  660 get a$:if a$<> chr$(13) then 660
OJ  670 end
MB  680 :
FK  690 rem begin the show
AD  700 :
MO  710 xin=-2*π      : rem  minimum value for x
MP  720 xax= 2*π      : rem  maximum value for x
BE  730 yin=-3.0      : rem  minimum value for y
AG  740 yax= 3.0      : rem  maximum value for y
BN  750 sx=160/(xax-xin)  : rem  scale in x direction
KN  760 sy=200/(yax-yin)  : rem  scale in y direction
PP  770 dx=(xax-xin)/160  : rem  set inc for 160 point plot
PO  780 pf=2          : rem initialize plotting flag
CO  790 rem  pf=0  . . . last point calculated was plotted o.k.
ML  800 rem  pf=1  . . . last point calculated was out of range
AE  810 rem  pf=2  . . . last point calculated was undefined
OP  820 x=xin - dx    : rem  initialize x
OL  830 sys hires,1,0,1,0,1
MA  840 sys dmode,1:sys selpc,1
LA  850 goto 190
```

VIC Parameters

Chris Zamara, Technical Editor
Program by Paul Higginbottom

C64 Video and Character Memory Allocation

The C64's flexible memory architecture and VIC–II video chip allow you to set up screen and character memory practically anywhere you want to, providing the ability to re–define characters and alter the system memory map to your own specifications. The problem is, it's not such a simple process — as usual, there are just enough complications to create confusion. This article, the enclosed program ("VICPARMS"), and the tables below are calculated to ease some of that confusion.

Fundamental Concepts

The VIC-II video chip in the C64 generates a video display based on data found within the 64K of memory in the computer. When in regular text mode (the default after power–up), there are two kinds of data needed to display characters on the screen: pointers which indicate which character is to be displayed in each screen position, and a number of bytes which serve as shape tables to describe what each character looks like.

The 1000 bytes which determine which characters are displayed on the screen are collectively known as screen memory, video memory, or more technically, the video matrix. The default location of the video matrix is at 0400 hex ($0400 or 1024 decimal), just below the BASIC text workspace, but that location can be changed at will.

Every one of the possible 256 values of each video matrix byte has a character associated with it. The shape of that character is defined by the contents of 8 bytes within character memory. Since there are 256 characters it follows that character memory must occupy 2048 bytes. On the 64, as with all Commodore machines, there are two character sets available: upper case/graphics, and upper/lowercase characters (these two sets are alternately selected with the shift/Commodore keys). Altogether then, there are 4096 bytes which define the shape of all 512 characters.

Character memory is normally found in ROM, at location $D000 (this is the same address range where the I/O is mapped, but either RAM or the character ROM can be mapped in instead). Like the video matrix, the start location of character memory (called the *character base*) can be changed. Of course, if you want to see legible characters on the screen, valid character definitions must exist in memory wherever the character base points. To accomplish that, the contents of the character ROM starting at $D000 can be copied into RAM wherever the new character base is. With the character definitions in RAM, you can customize the character set to your own specifications — that is the prime reason for changing the location of the character base.

Allocating video and character data to different areas of memory means pointing the VIC-II video chip to the start of the desired memory addresses by changing one of its registers. In the case of the video matrix, it also involves setting a pointer so that the kernal (operating system) knows where the screen is located in memory.

The start address of video or character memory can't just be anywhere; the video matrix must start on a 1K boundary, and the character base has to start on a 2K boundary. In other words, there are 64 possible places where screen memory can go, and 32 possible places where the character base can start. Table 1 contains a list of all locations where screen memory can go, and table 2 gives the 32 possible character base locations.

VIC Chip Banks

In practice, selecting the memory areas for the video matrix and character base is complicated by the fact that the VIC chip has only 14 address lines and as such can access only 16K of memory. To allow full access of all 64K of RAM in the 64, the most significant two address bits from the VIC chip are provided by two I/O lines (port A of CIA2). What that means to the programmer is that the video matrix and character base must lie within the same 16K boundary, since the VIC chip can only access one 16K "bank" at a time.

Note the way the video tables below are organized into four columns; each column lists the possible memory addresses in each 16K bank. The default bank is 0, since the video matrix begins at $0400. How then is it possible to access the character ROM, which resides at $D000, apparently in bank 3?

Character ROM Images

That feat is accomplished by a little trick in the 64's hardware. The character ROM appears at $D000 only to the CPU — as far as the VIC chip is concerned (remember it can only access up to 16K), it fetches its usual character definitions from $1000. However, the character definitions from $D000 only appear to

the VIC chip at $1000 *in banks 0 and 2*. So if you were to put character definitions in RAM at $1000 or $5000, the VIC chip wouldn't see them; it would still see the usual ROM character table. On the other hand, in banks 1 and 3 the VIC chip can get its character definitions out of RAM from any address within the 16K block, including the RAM at $D000 "under" the character ROM. To put it simply, the VIC chip will recognize character definitions from RAM starting at any 2K block in memory except at $1000, $1800, $5000 or $5800. Pointing the character base at any of these locations will result in displaying the default ROM characters.

Selecting One of Two Character Sets

As previously mentioned, there are two character sets defined in the ROM of the 64. The 2K of memory from $D000 to $D7FF contains the definitions for the upper-case/graphics character set which appears as the default after a reset or a RESTORE. The character definitions for the upper/lowercase character set (which can be selected by simultaneously pressing the SHIFT and "Commodore" keys) are located in the next 2K of ROM, from $D800 to $DFFF. To select one character set or the other, the kernal just flips the least significant bit of the character base pointer in the VIC chip to select an odd or even 2K boundary. This is worth noting because it means that the only way to set up two character sets and switch between them with the shift/Commodore key combination is to start the character definitions on a 4K boundary.

Specifics: How to Change the Pointers

Both the video matrix and character base addresses within a given bank are defined with VIC register 24, which is at location $D018 (53272 decimal). The video matrix start location (one of 16 addresses within the bank) is defined by bits 4-7, and bits 1-3 give the character base. The least significant bit is unused. In tables 1 and 2, the bit patterns necessary to select a given memory address appear in the rightmost column.

The bank select bits are accessed through location $DD00, which controls port A of CIA1, one of the two 6526 I/O chips. Bits 0 and 1 of this location control the state of the most significant bits of the VIC chip address bus. The bits are inverted so that both bits set to 0 selects bank 3, and both 1 selects bank 0. The state of these bits necessary to select each bank is given in the top row of tables 1 and 2.

It was mentioned earlier that relocating the video matrix also involves changing an operating system pointer. This is so that BASIC will put characters in the right place when PRINTing to the screen. Location $0288 (648 decimal) contains the page number of the video matrix in memory — it's normally set to 4, since the screen is at $0400. As another example, to relocate a screen to $4000 this byte would have to be changed to $40 (64 decimal).

It's worth mentioning here that there are other important memory areas to the VIC chip besides screen and character memory. There is colour memory, which appears at $D800 to the CPU — it is fixed, and can't be put anywhere else. In fact, colour memory isn't a part of the 64K memory map at all, but exists in the form of a lone 1K by 4 bit chip on the 64's circuit board.

Another source of VIC chip display data that is moveable in memory is the high-resolution screen. This only applies in hi-res mode, where 8000 bytes are required for the screen. That means that there are two possible places in each bank where a hi-res screen can be located. As indicated in Table 3, bit 3 of location $D018 controls whether the hi-res screen comes from the lower or upper 8K of a given bank.

The Easy Way Out: VICPARMS

If it sounds like a real pain to re-define the character set and re-allocate video memory, try the program in listing 1. VICPARMS was originally written by Paul Higginbottom from Commodore, and it does the dirty work for you. VICPARMS will ask you where you want to put screen memory (the video matrix), and where you want character memory to come from. You must supply these addresses in hexadecimal. Given this information, VICPARMS will determine the correct values to store in the necessary locations, and whether or not the character set must be transferred from ROM to RAM. Before going ahead, it will display the steps it's going to take, and ask if it should proceed. Giving the reply "y" at this point will initiate the action. If the character set must be transferred, it will take a few seconds before the process completes.

When using VICPARMS, you have to keep in mind the information in table 1 and 2. You can only put screen memory or character memory at one of the locations designated in the table. Also, if you put screen memory and character memory close enough together so that they'll overlap, there's going to be problems. In short, VICPARMS works on a user-beware philosophy, but it does the job with a minimum amount of code; it's a programmer's utility, after all.

When VICPARMS transfers character memory to RAM, it copies the 2K of character definitions which appear in ROM starting at $D000. This is the character set which defines the upper/case graphics characters. If you want the alternate upper/lowercase character set, change the variable 'cset' in line 100 from 0 to 1. Alternatively, you could change the program so that it copies both sets at once: change the '2047' in line 450 to '4095'.

Once you've relocated the character set to RAM, you can redefine the characters by simply POKEing about, with one exception: if character memory has been moved to $D000 (the RAM under the character ROM and I/O) and the screen is somewhere else in bank 3 ($C000 for example), the only way to change the RAM to redefine the characters is to "map out" the I/O. The IRQs will have to be disabled when doing this.

But before getting too deeply involved in a discussion about memory management on the 64 (which could easily fill another article), let's leave it at that.

Table 1: Video Matrix

Bank Select Register = $DD00

Bank 0	Bank 1	Bank 2	Bank 3	
xxxxxx11	xxxxxx10	xxxxxx01	xxxxxx00	
Address Relative to CPU				$D018 Contents
0000	4000	8000	C000	0000xxxx
0400	4400	8400	C400	0001xxxx
0800	4800	8800	C800	0010xxxx
0C00	4C00	8C00	CC00	0011xxxx
1000	5000	9000	D000	0100xxxx
1400	5400	9400	D400	0101xxxx
1800	5800	9800	D800	0110xxxx
1C00	5C00	9C00	DC00	0111xxxx
2000	6000	A000	E000	1000xxxx
2400	6400	A400	E400	1001xxxx
2800	6800	A800	E800	1010xxxx
2C00	6C00	AC00	EC00	1011xxxx
3000	7000	B000	F000	1100xxxx
3400	7400	B400	F400	1101xxxx
3800	7800	B800	F800	1110xxxx
3C00	7C00	BC00	FC00	1111xxxx

('x' means 'don't care')

Table 2: Character Base

Bank Select Register = $DD00

Bank 0	Bank 1	Bank 2	Bank 3	
xxxxxx11	xxxxxx10	xxxxxx01	xxxxxx00	
Address Relative to CPU				$D018 Contents
0000	4000	8000	C000	xxxx000x
0800	4800	8800	C800	xxxx001x
1000(1)	5000	9000(1)	D000	xxxx010x
1800(2)	5800	9800(2)	D800	xxxx011x
2000	6000	A000	E000	xxxx100x
2800	6800	A800	E800	xxxx101x
3000	7000	B000	F000	xxxx110x
3800	7800	B800	F800	xxxx111x

(1) ROM image from $D000 (upper case/graphics) appears here
(2) ROM image from $D800 (upper/lower case) appears here

Note: Bit 1 of $D018 is toggled by the shift/Commodore keys.

Table 3: Bit Map Memory

Bank Select Register = $DD00

Bank 0	Bank 1	Bank 2	Bank 3	
xxxxxx11	xxxxxx10	xxxxxx01	xxxxxx00	
Address Relative to CPU				$D018 Contents
0000	4000	8000	C000	xxxx0xxx
2000	6000	A000	E000	xxxx1xxx

Listing 1: VICPARMS

```
JF  10 rem* "vicparms"
FL  11 rem* this program allows you to
LA  12 rem* put your screen memory
FJ  13 rem* and character memory
AD  14 rem* in any of the possible locations.
BJ  15 rem* it will transfer the character
EC  16 rem* set from rom to ram if necessary
DA  19 rem program by paul higginbottom
II  20 :
GP  100 cset=1:rem transfer: 0=upper/graphics,
        1=upper/lowercase
PK  110 print "(entries in hexadecimal)"
IN  120 a$="screen addr":gosub 540:sc=v
HG  130 a$="  char.addr":gosub 540:ca=v
GK  140 b=int(sc/16384) :rem screen bank #
II  150 a=int(ca/16384) :rem char bank #
FO  160 sv=sc-b*16384   :rem screen address
JG  170 cv=ca-a*16384   :rem char address
AK  180 if sv=cv goto 510
DO  190 rem error if screen and chars at same place
ID  200 if ((b=1)or(b=3)) and (a<>b) goto 510
AD  210 rem error if in different no-image banks
FP  220 vic=13*4096     :rem vic chip address
MD  230 c1=13*4096+12*256 :rem cia chip 1
PE  240 c2=13*4096+13*256 :rem cia chip 2
IL  250 p=int(cv/2048)*2+int(sv/1024)*16:if p<0 then 510
FF  260 rem char, screen loc in lo, hi nybble
HO  270 q=int(sc/256):if q<0 then 510
CB  280 rem q is screen memory page for kernal
AF  290 print "poke vic+24,"p
BJ  300 print "poke 648,"q
LE  310 r=(peek(c2)and252)or(3-b)
PK  320 print "poke cia2,"r
DG  330 if ((b=0)or(b=2)) and (((pand14)=4)or((pand14=6)))
        goto 350
OO  340 print "you need to move the character set to:"
        :print ca
IC  350 print:input "shall i do this";a$
DA  360 if left$(a$,1)<>"y" then end
PI  370 poke vic+24,p
PJ  380 poke 648,q
FB  390 poke c2,r
AL  400 if ((b=0)or(b=2)) and (((pand14)=4)or((pand14=6)))
        goto 480
CI  410 rem transfer character set if necessary
OJ  420 poke c1+14,peek(c1+14)and254:rem turn off irq's
CO  430 poke 1,peek(1)and251:rem see chars
LF  440 rom=vic+cset*2048  :rem character rom
LF  450 for i=0to2047:poke ca+i,peek(rom+i):next
        :rem move chars
BM  460 poke 1,peek(1)or4:rem see i/o again
GA  470 poke c1+14,peek(c1+14)or1: rem enable irq again
EC  480 print chr$(147);
KO  490 end
IG  500 :
NH  510 print "?illegal parameters":end
MH  520 :
JI  530 rem* input/convert subroutine*
MI  540 print a$;:a$=""":v=0:input a$
MD  550 if a$="" goto 540
BJ  560 for i=1 to len(a$):a=asc(mid$(a$,i))-48
        :a=a+7*(a>9):v=v*16+a:next
GF  570 return
```

BIGPRINT

Program by Allen R. Mulvey, Fulton, New York
Presented by Chris Zamara

A True Proportion High-Res Printer Dump Utility



Remember PICPRINT in the "Protection and Piracy" issue (Vol 5, issue 03)? Well, this one is better! In the true spirit of program evolution (of which this magazine is a firm believer), a reader took a given program, added a dash of brilliance, and came up with a winner.

To refresh your memory, PICPRINT allowed you to view a high resolution picture in memory (at 2000 hex), and print the picture to a Star Gemini or similar printer, using the 64's function keys. The picture could be printed in a "wide mode" which s t r e t c h e d it out across the page; a wierd and perhaps less than useful feature. Alas, even the "normal mode" printout served to stretch the picture slightly, since the aspect ratio (pixel width vs. height) of the printer differs from that of the screen.

The good news: like Brainstorming, wierd concepts of no intrinsic value can spark thought. (Light bulb suddenly appears in thought bubble over man's head - Eureka!.) Which brings us back to BIGPRINT. When a "wide" picture dump is initiated (via the F3 key), the resulting picture takes up about the width of a standard printer page, but its height is also increased, so that the overall proportions match up quite well to those of the screen. Specifically, the big printout is 50 percent wider than the normal size and twice as high. Now why didn't I think of that?

BIGPRINT retains the option of printing a "normal" sized picture, which it does in the same manner as PICPRINT. Some pictures are designed to look right on a printer rather than the screen, and look best printed in normal mode.

After BIGPRINT is installed with SYS 49152, F7 will toggle high-res/ text mode, F3 will print the picture in normal size, and F5 will give the big print. The stop key can be used to halt a picture in progress at the end of a line.

The BASIC loader follows. Have fun with it, and keep those brain-storms coming in!

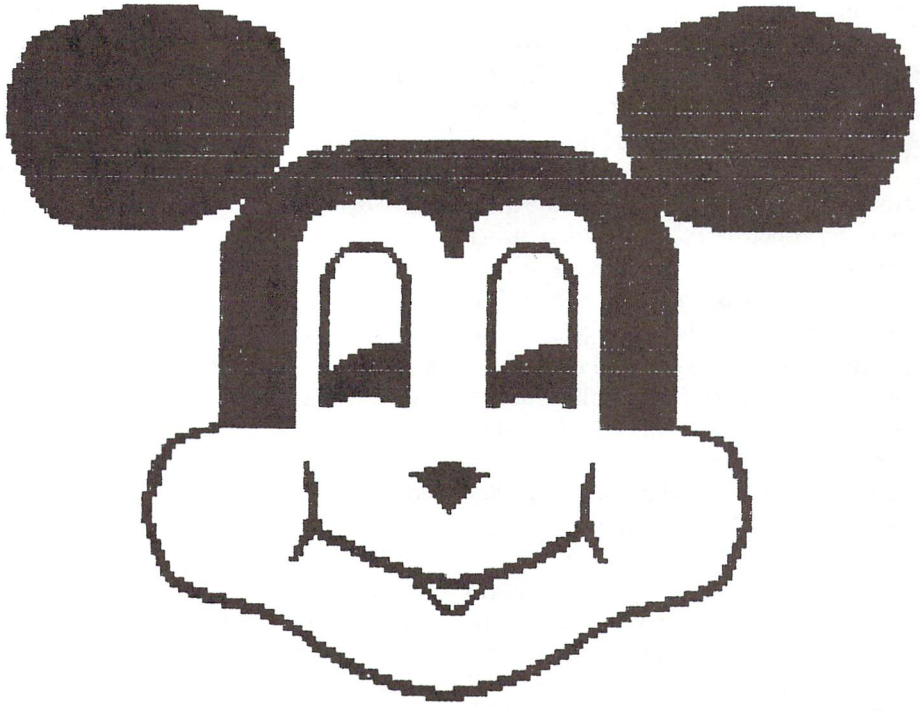
Notes:

1) Change the 76 in line 1030 to 121 to take advantage of the double speed print mode of the Star Gemini-10x printer.

2) The 4 in line 1280 is the secondary address for the Cardco interface, and selects graphics mode with auto-linefeeds. You may have to use the no-linefeed graphics mode (5), depending on the DIP switch settings on your printer.

```
CJ  10 rem* data loader for "bigprint"
CA  15 rem* for star gemini printer and
       cardco interface
LI  20 cs=0
JM  30 for i=49152 to 49639:read a:poke i,a
       :cs=cs+a:next i
MJ  40 :
PF  50 if cs<>58931 thenprint"**** error in
       data statements ****":end
JC  60 sys 49152
GE  70 end
EM  80 :
CD  1000 data  76,  46, 192,   0,   0,   0
BH  1010 data   0,   0,   7, 100, 128,  64
MD  1020 data  32,  16,   8,   4,   2,   1
JI  1030 data  27,  76, 192,   3,  27,  75
AC  1040 data  64,   1,  64,   0,  64,   0
MI  1050 data  64,  64,   0,  64,   0,  64
OJ  1060 data  64,   0,  64,  64,   0,  64
PG  1070 data  64,  64,   0,   0, 120, 169
PK  1080 data  59, 141,  20,   3, 169, 192
LJ  1090 data 141,  21,   3,  88,  96, 165
FD  1100 data 197, 201,  64, 208,   8, 169
BM  1110 data   0, 141,   4, 192,  76,  49
HA  1120 data 234, 173,   4, 192, 208,  43
IL  1130 data 169,   1, 141,   4, 192, 165
NP  1140 data 197, 201,   3, 240,  35, 201
LI  1150 data   6, 208,  16, 169,   3, 141
LB  1160 data   7, 192, 141,  45, 192, 169
FL  1170 data   0, 141,   5, 192,  76, 143
```

```
PO 1180 data 192, 201,   5, 208,   8, 169
LM 1190 data   0, 141,   7, 192,  76, 143
OI 1200 data 192,  76,  49, 234, 173,  17
GF 1210 data 208,  73,  32, 141,  17, 208
BD 1220 data 173,  24, 208,  73,   8, 141
DK 1230 data  24, 208,  76,  49, 234, 169
EB 1240 data   0, 141,   3, 192, 173, 141
KI 1250 data   2,  41,   4, 240,   5, 169
IC 1260 data 255, 141,   3, 192, 173,   9
LA 1270 data 192,  32, 195, 255, 173,   9
CB 1280 data 192, 162,   4, 160,   4,  32
NM 1290 data 186, 255, 169,   0,  32, 189
HB 1300 data 255,  32, 192, 255, 174,   9
NA 1310 data 192,  32, 201, 255, 169,   0
CF 1320 data 133, 251, 169,  32, 133, 252
FI 1330 data 169,  27,  32, 210, 255, 169
BN 1340 data  51,  32, 210, 255, 169,  16
CL 1350 data  32, 210, 255, 169,  25, 141
FB 1360 data  42, 192, 173,   5, 192, 240
BP 1370 data  13,  56, 165, 251, 233,  64
DL 1380 data 133, 251, 165, 252, 233,   1
EC 1390 data 133, 252, 162,   0, 189,  18
DN 1400 data 192, 172,   7, 192, 208,   3
CN 1410 data 189,  22, 192,  32, 210, 255
DN 1420 data 232, 224,   4, 208, 237, 169
EN 1430 data  40, 141,  43, 192, 169,   0
LE 1440 data 162,   7, 157,  34, 192, 157
JA 1450 data  26, 192, 202,  16, 247, 162
EO 1460 data   0, 160,   0, 177, 251, 141
NJ 1470 data  44, 192, 230, 251, 208,   2
OE 1480 data 230, 252, 173,  44, 192,  57
EC 1490 data  10, 192, 240,   9, 185,  34
DJ 1500 data 192,  29,  10, 192, 153,  34
CE 1510 data 192, 200, 192,   8, 208, 234
MC 1520 data 232, 224,   8, 208, 216, 173
OA 1530 data   7, 192, 240,  68, 169,   7
FO 1540 data 141,  45, 192, 174,   5, 192
JG 1550 data 189,   7, 192, 141,   6, 192
NL 1560 data 160,   7, 185,  34, 192, 174
CG 1570 data   6, 192,  61,  10, 192, 240
LP 1580 data  16, 185,  26, 192, 174,  45
KO 1590 data 192,  29,  10, 192, 202,  29
HJ 1600 data  10, 192, 153,  26, 192, 136
GB 1610 data  16, 226, 206,   6, 192, 206
MC 1620 data  45, 192, 206,  45, 192,  16
EA 1630 data 213, 160,   7, 185,  26, 192
IO 1640 data 153,  34, 192, 136,  16, 247
AM 1650 data 162,   0, 189,  34, 192,  77
GP 1660 data   3, 192,  32, 210, 255, 172
GH 1670 data   7, 192, 240,   6,  32, 210
LL 1680 data 255,  32, 210, 255, 232, 224
DD 1690 data   8, 208, 231,  76, 167, 193
GB 1700 data  76,   6, 193, 206,  43, 192
HO 1710 data 208, 248, 169,  13,  32, 210
MO 1720 data 255, 165, 197, 201,  63, 240
NE 1730 data  27, 208,   3,  76, 218, 192
OF 1740 data 173,   7, 192, 240,  12, 173
BA 1750 data   5, 192,  73,   1,  41,   1
CB 1760 data 141,   5, 192, 208, 236, 206
LI 1770 data  42, 192, 208, 231, 169,  27
AN 1780 data  32, 210, 255, 169,  64,  32
CL 1790 data 210, 255, 173,   9, 192,  32
JJ 1800 data 195, 255,  32, 204, 255,  76
BJ 1810 data  49, 234
```



"Mickey" by Coy V. Ison
created with 'Picture Perfect', printed with BIGPRINT

Two Short Sprite Editors

Chris Zamara, Technical Editor

The good thing about sprite editors is they make it easy to design your own sprites. The bad thing about sprite editors is if you don't have one, you have to type one in.

Of course, that's only bad news if the sprite editor program is very long. And that will serve as introduction to the programs presented below.

The sprite editors below work in different ways, but they share the same basic philosophy: why have oodles of code to allow cursor movement, character printing and deletion, etc. when the built-in BASIC editor will do all that just fine, thank you?

Sprite Editor Number 1

The first one is the shorter of the two, and a bit boring perhaps, but it works. The idea behind this one is to have the sprite definition contained within DATA statements. That way you can edit all the DATA statements on the screen, press RETURN over each one, then RUN the program to define and display the sprite. A bonus of doing it this way is that saving the program once a sprite is so defined also saves the sprite definition.

Look at listing 1. Where are the data statements? The program shares a trait with a monster robot in an old Japanese sci-fi movie: it can "program itself to enlarge" (honest, that's what the robot did in the movie). Enter the program, then RUN 300. Lines 1000 to 1020 will be added to the program, and will be listed for your convenience. Each period between the quotes in the DATA statements represents one pixel in the sprite. Change a dot to anything else (I like asterisks) to set the corresponding sprite pixel. You'll have to press RETURN over each line to enter it (the program sets auto-repeat on all keys so just start at line 1000 and hold down the RETURN key to enter all lines).

When you want to see the sprite, RUN the program; the newly defined sprite will appear at the top left corner of the screen. To continue editing, LIST1000- or just LIST to display the current DATA statements. The sprite is defined at sprite page 13 which is location 832. Thus, the bytes stored from 832 to 895 comprise the sprite definition, and may be written to a disk file or otherwise saved in lieu of saving the sprite editor program with the DATA statements. Sure, it's primitive, but it works, and the program is only 20 lines long. The main inconvenience is that you can't see the sprite dynamically change as each point is changed; you have to RUN the program each time to see the new sprite, and the re-definition process isn't quick. Enter Sprite Editor #2.

Sprite Editor Number 2

This one is a bit longer, but much more exciting. The sprite definition process occurs dynamically, as you plot or erase each point. It can be fun to use, since the routine is interrupt-driven and you keep complete control over the machine while you define the sprite by putting asterisks on the screen. RUN it (leaving out the opening REMs if you wish), and after a few seconds, an array of dots will be printed starting at the top left of the screen. After the dots appear, the computer returns to its normal state – you are free to enter commands, edit or LIST a program, whatever. But try changing some of the dots to asterisks. As asterisks are entered in any of the indicated screen locations, the sprite is instantly updated and displayed at the right of the screen. Unlike program one, you shouldn't press RETURN over each line or you'll get a syntax error - use SHIFT/RETURN instead. The dots which are printed aren't really important at all, but they serve as a guide to the boundary of the sprite definition area (24 characters across by 21 down).

Remember, once the program has been RUN, any change in the sprite definition field on the screen will affect the sprite. If you were to LIST the program for example, you'd see the a funny looking sprite form, and then appear to "scroll" upwards with the program listing. As the asterisks in the program listing move across the sprite definition field, the sprite continuously changes to reflect the new asterisk pattern. Of course, listing is slowed down considerably, since the interrupts are stealing so much time.

When you've played around enough to come up with a sprite you're happy with, you can disable the interrupt-driven sprite program and print out the 64 bytes defining the sprite by entering CONT. This command appears below the dots which are printed out, so you can just move the cursor over the CONT and press RETURN. After the values have been printed out, the interrupt routine can be re-enabled with SYS 49152, and disabled with SYS 49165.

Both programs can be embellished to give more features, for example multi-colour sprite capability. Even if the programs grow to the point of belying their calling in life (being small), it doesn't matter. Somehow, it seems easier to write a huge program than to type someone else's program in.

Listing for Sprite Editor #1

```
KJ 100 rem* simple sprite designer *
NP 110 sp=13  :rem sprite page
FD 120 s=sp*64:rem sprite position
KL 130 poke2040,sp
OC 140 v=53248:pokev+21,1:rem display mob
PJ 150 pokev+39,1:pokev,25:pokev+1,51
PJ 160 fori=0to7:e(7-i)=2↑i:next
LJ 165 poke 650,128: rem repeat all keys
BG 170 :rem convert data to mob
BK 180 fori=1to21:reada$:forj=1to24step8
EM 190 v=0:fork=0to7:ifmid$(a$,j+k,1)<>"."
       then v=v+e(k)
PC 200 next k:poke s,v:s=s+1:nextj,i
CN 210 end
BN 220 :rem add data statements
IE 300 ifa>20thenlist1000-
BD 310 print chr$(147)1000+a"data";
CE 315 print chr$(34)"....."chr$(34)
FI 317 rem 24 periods
CN 320 a=a+1:print"a=" a":goto300"
DI 330 poke198,3:poke631,19:poke632,13:poke633,13
EF 340 end
```

Listing for Sprite Editor #2

```
OA 100 rem *  >> easy sprite editor <<  *
OJ 110 rem * -use normal editor to draw. *
CB 120 rem *   -use "*" to plot points.  *
LI 130 rem * -press return over "cont:" *
BN 140 rem *     to print sprite values.  *
OE 150 rem *         aug84 --cz--         *
EB 160 :
HG 170 fori=49152to49278:reada:pokei,a:next
GE 210 :
EN 220 rem * display sprite #13 (at 832) *
IP 230 v=13*4096
DI 240 poke2040,13
DJ 250 poke v,30:poke v+1,100:poke v+16,1
HL 260 poke v+21,1: poke v+39,1
CI 270 :
ME 280 rem * print design grid *
KH 290 print chr$(147);:fori=1to21
IC 295 print"....."  :next i
AH 296 rem 24 periods
BM 300 print"cont:"
AK 310 sys49152: rem * enable sprite draw
AE 320 end
EI 330 rem * "cont" executes the following
OJ 340 sys49165: rem * disable sprite draw
HG 350 print chr$(147)
OI 360 rem * print sprite values *
EG 370 fori=832to895:printpeek(i)",",:next
MH 380 end
DD 1000 data 120, 169,  37, 141,  20,   3
JG 1010 data 169, 192, 141,  21,   3,  88
KN 1020 data  96, 120, 169,  49, 141,  20
BP 1030 data   3, 169, 234, 141,  21,   3
OF 1040 data  88,  96, 128,  64,  32,  16
DE 1050 data   8,   4,   2,   1,   0,   0
EH 1060 data   3, 169,   0, 133, 251, 169
BF 1070 data   4, 133, 252, 162,   0, 169
DE 1080 data   3, 141,  35, 192,  32, 101
KO 1090 data 192, 173,  34, 192, 157,  64
BI 1100 data   3, 169,   8, 141,  36, 192
EM 1110 data 206,  35, 192, 208,  10, 169
NH 1120 data   3, 141,  35, 192, 169,  24
BP 1130 data 141,  36, 192, 165, 251,  24
AL 1140 data 109,  36, 192, 133, 251, 144
CO 1150 data   2, 230, 252, 232, 224,  64
OO 1160 data 208, 210,  76,  49, 234, 169
CH 1170 data   0, 141,  34, 192, 160,   7
EH 1180 data 177, 251, 201,  42, 208,   9
CH 1190 data 185,  26, 192,  13,  34, 192
LC 1200 data 141,  34, 192, 136,  16, 238
DE 1210 data  96
```

A List Scrolling Routine For The C64

Darren James Spruyt
Gravenhurst, Ontario

Dig out program lines from beyond the screen limits !

This program allows you to scroll both forwards and backwards through a BASIC program on the screen using the cursor up/down keys. It saves countless typings of 'LIST' and having to press the RUN/STOP key at the correct time.

The operation is very simple: after you have run the BASIC loader program (Listing 1), give the enabling 'SYS.' To scroll backwards through the listing, place the cursor on the top row of the screen and press cursor up. If the screen has no numbers along the left hand side, the listing will start from the very last line of the program. Conversely if you are at the bottom of the screen (with no line numbers on the screen) and press cursor down, the listing will start with the first line in the program.

If line numbers appear on the screen, upwards scrolling will list the line previous to the number closest to the top of the screen, and if scrolling down, the line number that follows the one closest to the bottom of the screen will be listed.

One note: Be sure to stop a real list before pressing the cursor down, or else nasty things may happen. That is, if you typed 'LIST 100-', press the RUN/STOP key before trying to scroll down the listing.

Listing 1 is the BASIC loader for LIST SCROLLER, while listing 2 is a disassembly of the program. A few notes: The vector at ($0300) is the 'ERROR MESSAGE LINK', and all errors pass through this vector as well as the list routine when it is finished. This vector is changed so that the program can regain control after a line has been listed or when an error occurs during line number fetches.

The programming method used could be denoted 'POST-INTERRUPT' - that is, when an interrupt occurs, a little pre-interrupt code is done, then the system interrupt code is performed (keyboard scan, updating TI$), and finally, control is returned to the post-interrupt routine before the interrupt returns. The advantage over a simple pre-interrupt technique is that the program can check if certain keys are pressed before the screen editor even knows they exist.

Listing 1: BASIC loader for scroll routine

```
CC  0 rem* data loader for "scroller" *
LI  20 cs=0
BF  30 for i=51456 to 52049:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
LJ  60 if cs<>73817then
        print"**** error in data statements ****":end
CI  70 rem sys 51456
AF  80 end
IN  100 :
IB  1000 data 120,173, 20,  3,141, 43
DC  1010 data 201,173, 21,  3,141, 44
EJ  1020 data 201,169,201,141, 21,  3
OE  1030 data 169, 30,141, 20,  3,169
PC  1040 data   0,141, 67,203, 88, 96
NH  1050 data  44, 67,203, 16,  3, 76
NP  1060 data 126,234,165,157, 48,  3
DJ  1070 data  76, 49,234,169,255,141
BP  1080 data  67,203,160,  5,104,153
LL  1090 data  69,203,136, 16,249,160
HB  1100 data  73,169,201, 72,152, 72
ID  1110 data   8, 72, 72, 72, 76, 49
JK  1120 data 234,164,198,240, 13,185
CF  1130 data 118,  2,162,  1,221, 60
AA  1140 data 203,240,  6,202, 16,248
LN  1150 data  76,159,202,165,214,221
AP  1160 data  62,203,240,  3, 76,159
BP  1170 data 202,142, 77,203, 32,178
CC  1180 data 202, 32, 25,203, 32,187
CA  1190 data 202,198,198,174, 77,203
GG  1200 data 224,  0,208,  3, 76,  4
MM  1210 data 202,162,255,232,224, 25
HC  1220 data 240, 11,181,217, 16,247
AC  1230 data  32,229,202,144,242,176
GG  1240 data   6,169,255,133, 20,133
KC  1250 data  21, 32, 19,166,164, 95
KF  1260 data 228, 44,208,  7,196, 43
HH  1270 data 208,  3, 76,156,202,202
DH  1280 data 228, 44,176,  5,166, 44
GH  1290 data 164, 43,136,134, 64,132
OG  1300 data  63,160,  0,177, 63,240
DL  1310 data   6,200,208,249, 76,156
KK  1320 data 202,200,177, 63,197, 95
OG  1330 data 208,243,200,177, 63,197
GO  1340 data  96,208,236,136,152, 24
AA  1350 data 101, 63,133, 95,165, 64
```

```
NF 1360 data 105,   0, 133,  96, 162,  24
IL 1370 data 181, 217,  48,   5, 162,  23
JM 1380 data  32, 255, 233, 162,   0,  32
JL 1390 data 104, 233, 165, 217,   9, 128
MK 1400 data 133, 217, 169,  39, 133, 213
MJ 1410 data 162,   1, 142, 146,   2, 202
BO 1420 data 134, 214,  32, 240, 233,  76
DF 1430 data  85, 202, 162,  25, 202,  48
CG 1440 data  11, 181, 217,  16, 249,  32
IA 1450 data 229, 202, 144, 244, 176,   9
DA 1460 data 169, 255, 133,  20, 133,  21
MG 1470 data  72, 208,  21, 224,  24, 208
FG 1480 data   4, 169,  13, 208, 245, 224
AI 1490 data  23, 208,   5, 232, 181, 217
NE 1500 data  16, 243, 169, 255, 208, 232
IB 1510 data 230,  20, 208,   2, 230,  21
LK 1520 data  32,  19, 166, 165, 218,  48
IE 1530 data   9,   9, 128, 133, 218, 162
GB 1540 data   1,  32, 255, 233, 162,  24
FB 1550 data 134, 214,  32, 240, 233, 104
KH 1560 data  48,   3,  32, 210, 255, 160
FF 1570 data   1, 177,  95, 240,  37, 141
DL 1580 data  64, 203, 169,   0, 145,  95
JP 1590 data 132,  15, 165,  95, 133,  63
LM 1600 data 165,  96, 133,  64, 169, 255
AO 1610 data 133,  20, 133,  21,  76, 215
KJ 1620 data 166, 224,  11, 240,  26, 173
FL 1630 data  64, 203, 160,   1, 145,  63
HP 1640 data 169,  13,  32, 210, 255,  32
FK 1650 data 202, 202,  32,  47, 203, 169
BK 1660 data   0, 141, 146,   2,  76, 159
FG 1670 data 202, 104, 104, 104, 104,  76
CM 1680 data 128, 202, 230, 198,  76, 133
NC 1690 data 202, 160,   0, 185,  69, 203
KG 1700 data  72, 200, 192,   6, 208, 247
IO 1710 data 169,   0, 141,  67, 203,  76
BA 1720 data 129, 234, 174, 135,   2, 165
HG 1730 data 206,  32,  19, 234,  96, 165
EP 1740 data 214, 141,  75, 203, 165, 211
CL 1750 data 141,  76, 203, 169,   0, 133
GE 1760 data 211,  96, 174,  75, 203, 134
DB 1770 data 214, 173,  76, 203, 133, 211
OL 1780 data  32, 108, 229, 160,   0, 132
EA 1790 data 207, 140, 140,   2, 200, 132
JD 1800 data 205, 140, 139,   2,  96,  32
BG 1810 data 240, 233, 160, 255, 200, 177
CI 1820 data 209, 192,  39, 240,  38, 201
OH 1830 data  32, 240, 245, 201,  48, 144
BN 1840 data  30, 201,  58, 176,  26, 152
PG 1850 data  24, 101, 209, 133, 122, 165
BK 1860 data 210, 105,   0, 133, 123,  32
BG 1870 data 121,   0, 142,  68, 203,  32
NN 1880 data 107, 169, 174,  68, 203,  56
KI 1890 data  36,  24,  96, 160,   1, 185
AJ 1900 data   0,   3, 153,  65, 203, 136
KH 1910 data  16, 247, 169, 117, 141,   0
NJ 1920 data   3, 169, 202, 141,   1,   3
AO 1930 data  96, 173,  65, 203, 141,   0
FH 1940 data   3, 173,  66, 203, 141,   1
LE 1950 data   3,  96,  17, 145,  24,   0
LB 1960 data  12, 139, 227,   0,  23, 173
KA 1970 data 215, 224, 137,  97,   0,  24
IB 1980 data   0,   0,  65,  65,  65,  65
```

Listing 2: Commented Disassembly

```
c900 sei          ;lockout irq's
c901 lda  $0314   ;copy old irq to
c904 sta  $c92b   ;use in the exit
c907 lda  $0315   ;routine
c90a sta  $c92c   ;
c90d lda  #$c9    ;set new irq vector
c90f  sta  $0315  ;
c912 lda  #$1e    ;
c914 sta  $0314   ;
c917 lda  #$00    ;set processing to
c919 sta  $cb43   ;non active mode
c91c cli          ;
c91d rts          ;
c91e bit  $cb43   ;processing active?
c921 bpl  $c926   ;no
c923 jmp  $ea7e   ;skip out quick
c926 lda  $9d     ;in immediate mode
c928 bmi  $c92d   ;yes
c92a jmp  $ea31   ;otherwise exit
c92d lda  #$ff    ;set processing to
c92f  sta  $cb43  ;the active state
c932 ldy  #$05    ;remove six bytes
c934 pla          ;from the stack,
c935 sta  $cb45,y ;put there
c938 dey          ;by the
c939 bpl  $c934   ;interrupt routine
c93b ldy  #$49    ;set up false irq
c93d lda  #$c9    ;return
c93f  pha         ;vector and put
c940 tya          ;it onto the
c941 pha          ;stack
c942 php          ;push false status
c943 pha          ;push 3 more
c944 pha          ;dummy values onto
c945 pha          ;the stack
c946 jmp  $ea31   ;cont with interrupt
                  ;control returns to
                  ;here when irq done
c949 ldy  $c6     ;get # chars in buf
c94b beq  $c95a   ;empty
c94d lda  $0276,y ;get last char
c950 ldx  #$01    ;
c952 cmp$cb3c,x   ;check against
c955 beq  $c95d   ;the wanted chars
c957 dex          ;
c958 bpl  $c952   ;
c95a jmp  $ca9f   ;no matches
c95d lda  $d6     ;cursor row
c95f  cmp$cb3e,x  ;check against row
c962 beq  $c967   ;needed to execute
c964 jmp  $ca9f   ;no match here
c967 stx  $cb4d   ;up/down flag
c96a jsr  $cab2   ;erase cursor
c96d jsr  $cb19   ;change ($0300)
c970 jsr  $cabb   ;save cursor pos
c973 dec  $c6     ;purge from buf
c975 ldx  $cb4d   ;type up/dwn
c978 cpx  #$00    ;down number
c97a bne  $c97f   ;not equal – up
c97c jmp  $ca04   ;jmp to cursor down
c97f  ldx  #$ff   ;start of cursor up
c981 inx          ;
c982 cpx  #$19    ;checked all lines
c984 beq  $c991   ;yes
c986 lda  $d9,x   ;get link
c988 bpl  $c981   ;line's linked retry
c98a jsr  $cae5   ;try to linenumber
c98d bcc  $c981   ;none present
c98f  bcs  $c997  ;skip over
c991 lda  #$ff    ;set for top
c993 sta  $14     ;most line
c995 sta  $15     ;to be found
c997 jsr  $a613   ;find line
c99a ldy  $5f     ;lo byte
c99c cpx  $2c     ;compare with Start
c99e bne  $c9a7   ;of basic hi
c9a0 cpy  $2b     ;compare with Start
c9a2 bne  $c9a7   ;of basic lo
c9a4 jmp  $ca9c   ;no previous lines
c9a7 dex          ;check x lower than
c9a8 cpx  $2c     ;start of basic
c9aa bcs  $c9b1   ;not at all
c9ac ldx  $2c     ;correct if lower
c9ae ldy  $2b     ;that start of basic
c9b0 dey          ;
c9b1 stx  $40     ;set up indirect
c9b3 sty  $3f     ;
c9b5 ldy  #$00    ;set to search
c9b7 lda  ($3f),y ;for a zero byte
c9b9 beq  $c9c1   ;meaning end of line
c9bb iny          ;no match
c9bc bne  $c9b7   ;try  for next
c9be jmp  $ca9c   ;no more chances
c9c1 iny          ;check the value
c9c2 lda  ($3f),y ;against the link lo
c9c4 cmp$5f       ;
c9c6 bne  $c9bb   ;no match, retry
c9c8 iny          ;
c9c9 lda  ($3f),y ;check value against
c9cb cmp$60       ;the link hi
c9cd bne  $c9bb   ;to match
c9cf  dey         ;
c9d0 tya          ;make ($5f)
c9d1 clc          ;match
c9d2 adc  $3f     ;($3f)
c9d4 sta  $5f     ;plus
c9d6 lda  $40     ;the y register
c9d8 adc  #$00    ;offset
c9da sta  $60     ;
c9dc ldx  #$18    ;check bottom line
c9de lda  $d9,x   ;for a link
c9e0 bmi  $c9e7   ;not linked
c9e2 ldx  #$17    ;erase second bottom
c9e4 jsr  $e9ff   ;elim half lines
c9e7 ldx  #$00    ;set to scroll down
c9e9 jsr  $e968   ;do it
c9ec lda  $d9     ;fix the top lines
c9ee ora  #$80    ;link
c9f0  sta  $d9    ;value
c9f2  lda  #$27   ;fix its length also
c9f4  sta  $d5    ;
c9f6  ldx  #$01   ;set insert mode
c9f8  stx  $0292  ;
c9fb  dex         ;set cursor to
c9fc  stx  $d6    ;the home
c9fe  jsr  $e9f0  ;position
ca01 jmp  $ca55   ;into main list
ca04 ldx  #$19    ;set to scan
ca06 dex          ;screen
ca07 bmi  $ca14   ;done all lines
ca09 lda  $d9,x   ;get link byte
ca0b bpl  $ca06   ;linked, retry
ca0d jsr  $cae5   ;search for line#
ca10 bcc  $ca06   ;not present
ca12 bcs  $ca1d   ;yes skip over
ca14 lda  #$ff    ;set for first line
ca16 sta  $14     ;
ca18 sta  $15     ;
ca1a pha          ;push a byte
ca1b bne  $ca32   ;jump over code
ca1d cpx  #$18    ;was line# on bottom
ca1f  bne  $ca25  ;no
ca21 lda  #$0d    ;put byte onto stack
ca23 bne  $ca1a   ;
ca25 cpx  #$17    ;was line# on 24th
ca27 bne  $ca2e   ;no skip over
ca29 inx          ;
ca2a lda  $d9,x   ;was bottom lineked
ca2c bpl  $ca21   ;yes,byte on stack
ca2e lda  #$ff    ;
ca30 bne  $ca1a   ;put byte onto stack
ca32 inc  $14     ;increase lnum by 1
ca34 bne  $ca38   ;
ca36 inc  $15     ;
ca38 jsr  $a613   ;find line number
ca3b lda  $da     ;second line (top)
ca3d bmi  $ca48   ;linked?
ca3f  ora  #$80   ;yes, unlick it
ca41 sta  $da     ;
ca43 ldx  #$01    ;erase the
ca45 jsr  $e9ff   ;the line
ca48 ldx  #$18    ;set cursor to
ca4a stx  $d6     ;bottom
ca4c jsr  $e9f0   ;of screen
ca4f  pla         ;pull a byte
ca50 bmi  $ca55   ;negative skip over
ca52 jsr  $ffd2   ;print the return
ca55 ldy  #$01    ;main list entry
ca57 lda  ($5f),y ;check linkage
ca59 beq  $ca80   ;end of lines
ca5b sta  $cb40   ;save link byte
ca5e lda  #$00    ;change link to fool
ca60 sta  ($5f),y ;the list routine
ca62 sty  $0f     ;set list quote flg
ca64 lda  $5f     ;copy ($5f) to
ca66 sta  $3f     ;($3f)
ca68 lda  $60     ;
ca6a sta  $40     ;
ca6c lda  #$ff    ;set line list top
ca6e sta  $14     ;to a maximum value
ca70 sta  $15     ;
ca72 jmp  $a6d7   ;into list routine
ca75 cpx  #$0b    ;means error from
ca77 beq  $ca93   ;eval fxd pnt num
ca79 lda  $cb40   ;fix up
ca7c ldy  #$01    ;the link that was
ca7e sta  ($3f),y ;changed earlier
ca80 lda  #$0d    ;print
ca82 jsr  $ffd2   ;a return
ca85 jsr  $caca
ca88 jsr  $cb2f
ca8b lda  #$00    ;turn off insert mde
ca8d sta  $0292   ;
ca90 jmp  $ca9f   ;into exit routine
ca93 pla          ;pull byte from
ca94 pla          ;fxd pnt routine
ca95 pla          ;and from find line
ca96 pla          ;routine
ca97 jmp  $ca80   ;into fix up rtne
ca9a inc  $c6     ;re–instate char
ca9c jmp  $ca85   ;enter fix up rtne
ca9f ldy  #$00    ;main exit rout
caa1 lda  $cb45,y ;push all the
caa4 pha          ;saved data
caa5 iny          ;back onto
caa6 cpy  #$06    ;the stack
caa8 bne  $caa1   ;
caaa lda  #$00    ;set processing to
caac sta  $cb43   ;null state
caaf  jmp  $ea81  ;and exit
cab2 ldx  $0287   ;erase cursor
cab5 lda  $ce     ;routine
cab7 jsr  $ea13   ;
caba rts          ;
cabb lda  $d6     ;save cursor row
cabd sta  $cb4b   ;
cac0 lda  $d3     ;save cursor column
cac2 sta  $cb4c   ;
cac5 lda  #$00    ;set column to
cac7 sta  $d3     ;a zero value
cac9 rts          ;
caca ldx  $cb4b   ;restore cursor row
cacd stx  $d6     ;
cacf  lda  $cb4c  ;restore cursor col
cad2 sta  $d3     ;
cad4 jsr  $e56c   ;fix cursor status
cad7 ldy  #$00    ;fix
cad9 sty  $cf     ;up all the delay
cadb sty  $028c   ;counters so
cade iny          ;that scrolling
cadf  sty  $cd    ;will occur
cae1 sty  $028b   ;much faster
cae4 rts          ;
cae5 jsr  $e9f0   ;set line
cae8 ldy  #$ff    ;
caea iny          ;
caeb lda  ($d1),y ;get char from line
caed cpy  #$27    ;at end of line
caef  beq  $cb17  ;yes
caf1  cmp#$20     ;a space?
caf3  beq $caea   ;yes
caf5  cmp#$30     ;under ascii '0
caf7  bcc $cb17   ;yes
caf9  cmp#$3a     ;over ascii '9
cafb  bcs  $cb17  ;yes
cafd tya          ;fix ($7a) to
cafe  clc         ;match ($d1)
caff  adc $d1     ;plus the
cb01 sta  $7a     ;index
cb03 lda  $d2     ;int the y
cb05 adc #$00     ;register
cb07 sta  $7b     ;
cb09 jsr  $0079   ;chrgot – set flags
cb0c stx  $cb44   ;save x – reg
cb0f  jsr  $a96b  ;eval number
cb12 ldx  $cb44   ;restore x val
cb15 sec          ;carry set = found
cb16 bit  $18     ;entry for carry clr
cb18 rts          ;
cb19 ldy  #$01    ;copy ($0300)
cb1b lda  $0300,y ;vector to memory
cb1e sta  $cb41,y ;
cb21 dey          ;continue
cb22 bpl  $cb1b   ;
cb24 lda  #$75    ;set new vector
cb26 sta  $0300   ;
cb29 lda  #$ca    ;
cb2b sta  $0301   ;
cb2e rts          ;
cb2f  lda  $cb41  ;restore the old
cb32 sta  $0300   ;($0300) vector
cb35 lda  $cb42   ;
cb38 sta  $0301   ;
cb3b rts          ;
cb3c              ;data cursor up/down
cb3e              ;cursor down line
cb3f              ;cursor up line
cb40              ;link byte save
cb41              ;storage for the
cb42              ;($0300) vector
cb43              ;processing status
cb44              ;x–reg temp storage
cb45              ;stack storage
cb47              ; (six bytes )
cb48              ;
cb49              ;
cb4b              ;cursor row
cb4c              ;cursor col value
cb4d              ;up/down flag
```

STP: Execute From A Sequential File

Chris Zamara, Technical Editor

STP stands for "Sequential To Program", as this utility was originally written to convert a sequential program listing into a BASIC program. That enables you to LIST a program to disk, edit it with a word processor, then merge it with a program currently in memory. And if you felt like it, you could even write a program from scratch with a word processor, save it as a sequential file, then use STP to turn it into a normal program.

As often happens with programs, STP turned out to be more useful than it was intended to be. STP turns a sequential file into a program by reading a disk file line by line (assuming carriage returns at the end of each line) and processing that line as if it were entered from the keyboard. So if the disk file happens to look like a BASIC program, then that program gets entered just as if you typed it in yourself. Consider, however, a file that looks like this:

```
load " game.ml " ,8,1
new
load " game.bas " ,8
10 rem
list
run
```

This file could be created like this:

```
open1,8,12, " 0:test file,s,w "


print#1, " load " chr$(34) " game.ml " chr$(34) " ,8,1 "
print#1, " new "
print#1, " load " chr$(34) " game.bas " chr$(34) " ,8 "
print#1, " 10 rem "
print#1, " list "
print#1, " run "

close#1
```

STP-ing that file would give the same results as typing in those commands from the keyboard: you'd LOAD some machine language, NEW, load a BASIC program, REM out line 10, LIST, then RUN the newly loaded program. STP can automate some procedures that you currently don't have a program to do. You could use it to boot up some programming utility packages, change some parameters, set up some defaults, colour your screen and border nicely, etc. If you're familiar with the Power 64 utility, you'll notice that STP does the same thing as Morepower's EXEC function.

STP is only 173 bytes long, and as listed here, it lives at $C000 or 49152 decimal. It may be convenient to assemble a version to lie at 828 in the cassette buffer (such a version is on the Transactor disk for this issue). To use it, just call it with a SYS followed by a comma and the filename of the sequential file to be STP'd. For example:

```
sys 49152, " test file
```

Any string expression is valid, for example:

```
10 input " filename " ;f$
20 sys 49152,f$
30 end
```

STP can be called from within a program, but will only become active AFTER the program ends. Pressing the STOP key will halt STP at the end of the line currently being read in.

For the curious, STP works through the 64's warm start link at $0302-0303. It routes this vector to the main routine, which reads in a line, prints it on the screen, puts a cursor-up and carriage return in the keyboard buffer, then jumps to the normal warm link entry point. The new line will be entered, and after it has been processed, control is again passed through the warm start vector and the process repeats until end-of-file is reached. At that point, the link vector is restored to its normal state.

The only other catch is the bit of trickery necessary to prevent the disk file from being closed after a BASIC line is entered or a NEW is performed. All open files are normally closed by a routine that is fortunately vectored through an "abort I/O vector" at $032C-032D. The vector points to $F32F, and the first thing that happens there is the accumulator is loaded with a zero to be stored in the number of open files variable at $0098. To keep one file open at all times, STP points the abort I/O vector to its own routine which loads the accumulator with one, then sneaks by the LDA instruction in the abort I/O routine, jumping to $F331 instead of $F32F. This effectively prevents the first file opened from ever closing. STP closes all open files before it opens the disk file so that the disk file will be the *first* open file.

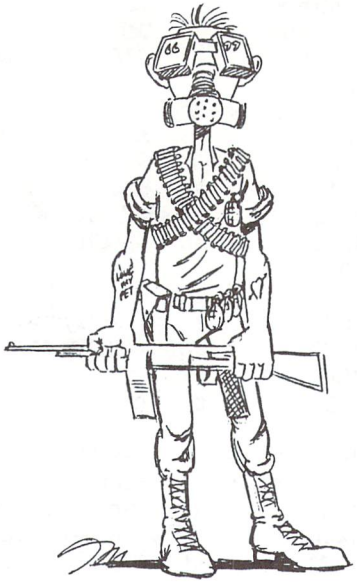
The BASIC Loader Program

```
AN 10 rem* data loader for "stp" *
LI 20 cs=0
PF 30 for i=49152 to 49325:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
OH 60 if cs<>21681 thenprint "**** error in data
      statments ****": end
PB 70 print "to run stp: sys49152,filename"
AF 80 end
IN 100 :
AF 1000 data  32, 253, 174,  32, 158, 173
AD 1010 data  32, 143, 173, 169, 100, 160
AM 1020 data 101,  32, 219, 182, 160,   0
PB 1030 data 177, 100,  72, 200, 177, 100
IG 1040 data  72, 200, 177, 100,  72, 173
AF 1050 data   2,   3, 141,  92, 192, 173
GH 1060 data   3,   3, 141,  93, 192, 169
CJ 1070 data  94, 141,   2,   3, 169, 192
MG 1080 data 141,   3,   3, 169,  87, 141
CI 1090 data  44,   3, 169, 192, 141,  45
LI 1100 data   3, 169, 127, 162,   8, 160
PI 1110 data  12,  32, 186, 255, 104, 168
JJ 1120 data 104, 170, 104,  32, 189, 255
CN 1130 data  32, 192, 255,  32, 183, 255
EM 1140 data 208,  28,  96, 169,   1,  76
EM 1150 data  49, 243,   0,   0, 162, 127
MA 1160 data  32, 198, 255,  32, 228, 255
EM 1170 data  32, 210, 255, 201,  13, 240
PM 1180 data  38,  32, 183, 255, 240, 241
CA 1190 data 169, 127,  32, 195, 255, 173
HP 1200 data  92, 192, 141,   2,   3, 173
MB 1210 data  93, 192, 141,   3,   3, 169
AD 1220 data  47, 141,  44,   3, 169, 243
NB 1230 data 141,  45,   3,  32, 204, 255
KL 1240 data 108,  92, 192, 169,   2, 133
FE 1250 data 198, 169, 145, 141, 119,   2
HJ 1260 data 169,  13, 141, 120,   2, 165
DG 1270 data 197, 201,  63, 240, 203,  76
OD 1280 data 141, 192,  32, 210, 255, 165
```

The source code for STP

```
HH 00    sys700                    ;pal 64
KF 1010 .opt oo
NO 1020     *     =     $c000
MH 1030 ;
CB 1040 ;"STP"
KM 1050 ;executes a sequential file
NK 1060 ;syntax is:
EB 1070 ;sys(addr),"filename"
OK 1080 ;
HG 1090 ;kernal entries:
EN 1100 chkin     =     $ffc6
NH 1110 chrout    =     $ffd2
FN 1120 clall     =     $ffe7
HL 1130 close     =     $ffc3
AG 1140 clrchn    =     $ffcc
EG 1150 getin     =     $ffe4
HM 1160 open      =     $ffc0
NH 1170 readst    =     $ffb7
FM 1180 setlfs    =     $ffba
IL 1190 setnam    =     $ffbd
GC 1200 ;
IF 1210 warmlnk   =     $0302
PI 1220 abortio   =     $032c
EE 1230 ;
LE 1240 lstx      =     $c5       ;last key pressed
PK 1250 ndx       =     $c6       ;# of keys pressed
ON 1260 keyd      =     $277      ;keyboard buffer
MG 1270 ;
MH 1280 ;get filename
MP 1290           jsr   $aefd     ;check for comma
IN 1300           jsr   $ad9e     ;evaluate expression
LI 1310           jsr   $ad8f     ;check for string
AF 1320           lda #$64: ldy #$65
BA 1330           jsr   $b6db     ;clean descriptor stack
CL 1340 ;
DO 1350           ldy #0:lda ($64),y: pha ;length
DI 1360           iny:lda ($64),y:pha    ;addr low
IH 1370           iny:lda ($64),y:pha    ;addr hi
KN 1380 ;
HL 1390           jsr   clall     ;close any previous files
OO 1400 ;
KP 1410 ;change warm start link
HL 1420 ;and abort i/o vector
FH 1430           lda warmlnk  : sta oldwarm
ME 1440           lda warmlnk+1: sta oldwarm+1
NJ 1450           lda #<newarm : sta warmlnk
KI 1460           lda #>newarm : sta warmlnk+1
BO 1470           lda #<newio  : sta abortio
AM 1480           lda #>newio  : sta abortio+1
IE 1490 ;
HN 1500 ;open disk file
JA 1510           lda #127: ldx #8: ldy #12
ML 1520           jsr   setlfs    ;open 127,8,12
EB 1530           pla:tay:pla:tax:pla
FM 1540           jsr   setnam
CI 1550           jsr   open
KP 1560           jsr readst: bne out0 ;disk error
OA 1570           rts
CK 1580 ;
KG 1590 newio     =     *         ;keep 1 file open
AK 1600           lda #1: jmp $f331
AM 1610 ;
FJ 1620 oldwarm .wor 0
EN 1630 ;
MG 1640 newarm    =     *
HI 1650 ;new warm start link points here
MG 1660           ldx #127: jsr chkin
FL 1670 nexbyt    =     *
NN 1680           jsr getin:jsr chrout
EN 1690           cmp #13: beq endlin
EP 1700           jsr readst: beq nexbyt
EC 1710 ;
ND 1720 out0      =     *
AN 1730           lda #127: jsr close
BP 1740           lda oldwarm  : sta warmlnk
AL 1750           lda oldwarm+1: sta warmlnk+1
NE 1760           lda #$2f: sta abortio
JA 1770           lda #$f3: sta abortio+1
LO 1780 out       =     *
FA 1790           jsr   clrchn
JK 1800           jmp   (oldwarm)
II 1810 ;
KN 1820 endlin    =     *
FA 1830           lda #2: sta ndx ;# keys in buffer
OI 1840           lda #145: sta keyd  ;crsr up. . .
KN 1850           lda #13 : sta keyd+1 ;& cr in kbuf
GL 1860           lda   lstx      ;key pressed
KC 1870           cmp #63: beq out0 ;check for stop
IA 1880           jmp   out
OD 1890 .end
```

Quote Killer



Gary Gunderson
Richmond, BC.

Quote killer was born on a bright sunny day as I lounged by the pool sipping a tall cool drink. Serendipity you say? You would be absolutely correct if you think serendipity has some relationship to a threat to your computer's well being.

My girlfriend had expressed interest in learning how to program my C64 (neat girlfriend, eh!, and no, she is not available). To that end, I had bought her a book for neophyte programmers into which she had plunged with enthusiasm. But alas, there she stood trembling with indignation and threatening to teach my machine how to do the swan dive.

All she could say was, "Dumb stupid quote mode!"

Thinking quickly while I gingerly plied my 64 from her hands, I promised her I would do something about it.

It is generally accepted that Commodore's on-screen editor is one of the best in the industry. The only major fault it has is the inability of the programmer to take the machine out of quote mode (or insert mode) without pressing the return key; the 64 lacks the handy ESC key of the 8032 machines. Consequently, you have to hit return and cursor back to the mistake to correct it. This doesn't present much of a problem, other than the inconvenience, to those of us who are used to it but for the novice, it is often a source of ongoing frustration.

The most obvious choice for a key to toggle quote killer was one of the function keys, and indeed F1 was the key I originally used. I immediately ran into problems with this when I started to design a program that made extensive use of the functon keys. At that time I realized that Commodore, in all its mystery and imagination, had included an apparently useless key on the keyboard. Tell me, when was the last time you used the back-arrow key for anything?

During this change of keys I decided to add a few features to quote killer which would make it much more useful to the advanced programmer. Quote killer now has the ability to force the machine into quote mode and to force a single character insert mode without opening any space in text. As an afterthought I also included the ability to force the repeat of any character on the keyboard when that particular key is held down. All the keystrokes that you need to know for the activation of quote killer's modes are included in the BASIC loader and will be displayed when the program is run.

Quote killer resides at the top of the cassette buffer and is therefore sensitive to any form of reset or tape operation. The only safe way to disconnect it is with RUN-STOP/RESTORE. SYS903 will reconnect it.

If for some reason you don't like my choice of the toggle key, you can alter it by changing the first data element of line 1050 to the keyboard matrix number of your choice. For example, a 4 in this location will make F1 the toggle. Remember, changing the BASIC loader in this way will change the checksum value, so change the checksum comparison accordingly or get rid of it altogether.

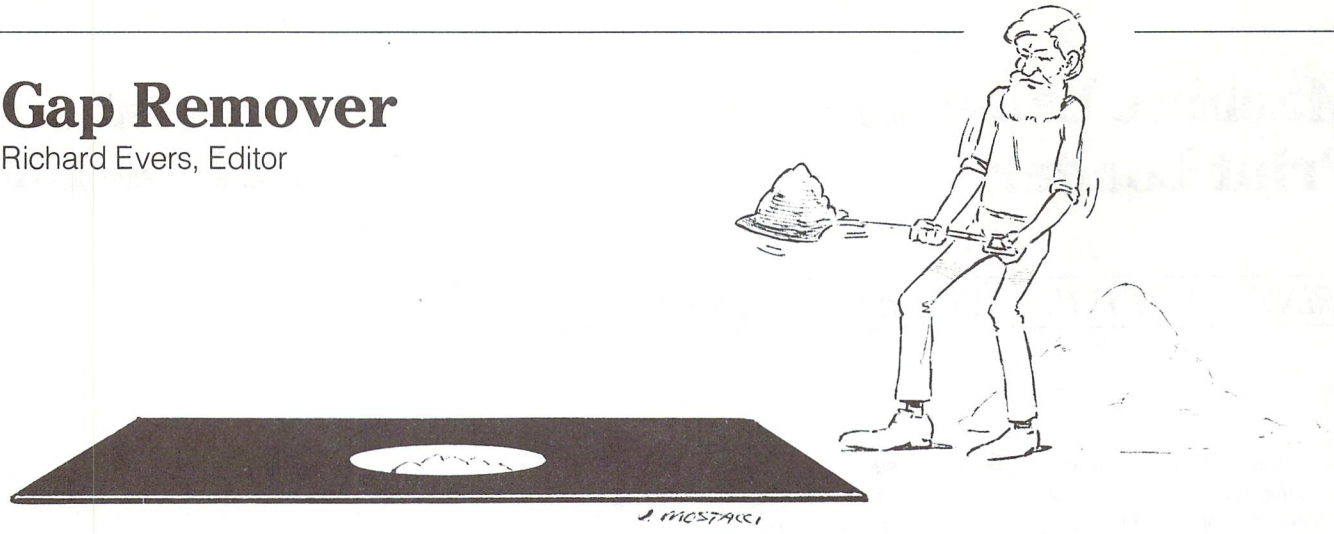
For those of you who are curious, quote killer demonstrates how to go about changing system vectors to suit your own needs. It is a very simple but powerful technique. The machine code is less than 100 bytes and should be very easy to follow.

By the way, having a girlfriend that likes to program is not always the most ideal situation. What do you do when she takes a piece of your immaculate code, shortens it by one third and makes it run 25 percent faster? Say "thank you"?

```
JH  1 rem  **********************
HC  2 rem  ***      quote killer      ***
GC  3 rem  ***           by           ***
EH  4 rem  ***  gary gunderson  ***
NH  5 rem  **********************
FN  10 fori=903to998:reada:cs=cs+a:poke i,a
DP  20 next i
LD  30 if cs<>11893 then print"** error in data
        statements!! **": end
OB  40 sys903:printchr$(147)"quote killer activated.":print
ED  50 print"  ←- kills quote mode
DF  60 print"      - kills insert mode
KB  70 print"      - kills repeat mode":print
IO  80 print"
FI  90 print"shifted ← -enables quote mode
AB  100 print"commodore ← -one char ins mode
LK  120 print"ctrl ← -enable repeat for all
KC  130 print"            keys":print
PI  150 print"run-stop/restore to deactivate
PG  160 print"sys903 to reactivate"
OB  170 :
LO  1000 data 173, 144,   2, 201,   3, 240,  86
BC  1010 data 173, 143,   2, 141, 248,   3, 173
KB  1020 data 144,   2, 141, 249,   3, 120, 169
ND  1030 data 167, 141, 143,   2, 169,   3, 141
JE  1040 data 144,   2,  88,  96, 165, 203, 201
FF  1050 data  57, 208,  55, 173, 141,   2, 201
CC  1060 data   1, 240,  22, 201,   2, 240,  25
CA  1070 data 201,   4, 240,  28, 169,   0, 133
JG  1080 data 212, 133, 216, 133, 199, 141, 138
KO  1090 data   2,  76, 224,   3, 169,   1, 133
GG  1100 data 212,  76, 224,   3, 169,   1, 133
BA  1110 data 216,  76, 224,   3, 169, 128, 141
NN  1120 data 138,   2, 234, 234, 234, 169,  64
NK  1130 data 133, 203, 108, 248,   3
```

Gap Remover

Richard Evers, Editor



Let's set the scenario. You have just SAVEd a file to diskette and, to set your mind at ease, performed a directory of the diskette. You view the entire directory only to discover that your file isn't there. You fall into great despair. . . a feeling that all your work has been in vain. With a sudden blast of inspiration, you look through the directory once again, very carefully this time. Within seconds your pulse is back to normal, but all is not right. Your file has mysteriously placed itself smack in the middle of the directory listing, thus falling into a prior scratched file's 'Black Hole'.

The situation is common, but often not as dramatic. If you have used your drive for longer than a few days, you already have accepted this rotten fact of life. Scratched files leave holes that future files trip into. This leads to disorganized directories, and strange delays when performing a passive catalog. A problem that now has a cure. Enter 'The Gap Remover'. Fire it up, answer a few important questions, and within a few minutes your directory will be clean and fresh, all extra spaces removed and placed at the very bottom. From then on, every file you SAVE will appear last in the directory.

Without burning up precious magazine space, the concept is as follows. The directory on your diskette is held on one track, track 18 for the 1541/2031/4040 drives, and track 39 for the 8050/8250 monsters. The first directory block is sector #1, with eight file entries per sector maximum. The first two bytes of every sector holds the track and sector of the next directory block. The sectors are spaced 3 apart, to allow for the drive to naturally reach it as it spins around. If the directory sector being read is the last one used, the track will read as a zero, and the sector will be $FF. The DOS realizes that there is no track zero, therefore the end has come. Simple so far.

The program reads in every entry from the directory track, keeping the link pointers in check as it goes along. If a non-existent file appears, one in which has been scratched, the file is flagged to the screen as showing as a gap, and its space is ignored. Every valid directory entry is kept in an array. When the entire directory has been read through, the packing process begins. The non-existent files have already been removed, therefore it is just a simple process of writing the packed entries back in. Once the process determines that the end of valid entries has occurred, null entries (32 x $00's) are written to the directory sector involved to fill in the extra space. When complete, you may end up having an extra directory sector or two if the packing required was large, but the directory will be OK. You can now SAVE files to your heart's content and never have to worry about them falling into the mysterious Black Holes of the DIR quadrant.

```
MD  10 rem save "0:gap fill.bas",8
AD  100 rem  rte/84 - gap filler for spaces left in your directory
HD  110 print "** directory gap filler - rte/84 **"
CK  120 print "drive type :"
BE  130 input "1) 1541/2031/4040 or 2) 8050/8250 ";dt: if dt<1
        or dt>2 then 130
GB  140 mtrk=18: if dt=2 then mtrk=39: rem assign correct
        directory track
NL  150 input "drive #  ";dr: if dr>1 then 150
EB  160 :
DP  170 dim dir$(223): rem ** max directory entries
DM  180 dim tl$(27), sl$(27): rem ** track/sector links
OF  190 trk=mtrk: sec=1: rem directory track and first sector to use
JK  200 ctr=0: lnk=0: z$=chr$(0)
IJ  210 rp$="": for x=0 to 29: rp$=rp$+chr$(0): next: rem
        ** replacement string
AF  220 :
IH  230 open 15,8,15: open 5,8,5,"#": rem command channel
        + buffer
EK  240 print#15,"u1:";5;dr;trk;sec: rem read dir entry into buffer
OG  250 :
AK  260 for x=0 to 255 step 32
CD  270 get#5,t$,s$: if x then 290: rem get correct links
BE  280 trk=asc(t$+z$): sec=asc(s$+z$): rem link t/s
CJ  290 for y=2 to 31: get#5,a$:dir$(ctr)=dir$(ctr)+chr$(asc(a$+z$))
        : next y
FE  300 a$=mid$(dir$(ctr),4,16): if asc(dir$(ctr)) then 320: rem if
        not scratched
GE  310 print a$ tab(17)"<< gap >>": dir$(ctr)="": goto 330
BF  320 print a$: ctr=ctr+1
GF  330 next x
IM  340 :
FE  350 tl$(lnk)=chr$(trk): sl$(lnk)=chr$(sec)
IM  360 if trk then lnk=lnk+1: goto 240: rem ok - go for more
GO  370 :
CC  380 rem ** directory completely loaded in - now time to pack **
II  390 trk=mtrk: sec=1: fin=0
KH  400 for x=0 to lnk: print#15,"b-p:";5;0
LH  410 ts$=tl$(x)+sl$(x): dd$="": rem link t/s
GH  420 for y=0 to 7: a$=dir$(fin): if fin=>ctr then a$=rp$
JO  430 dd$=dd$+a$: if y<7 then dd$=dd$+chr$(0)+chr$(0)
        : rem wasted 2 bytes/entry
DM  440 fin=fin+1: next y
JE  450 print#5,ts$;dd$;: print#15,"u2:";5;dr;trk;sec;: rem fill
        buffer then write
PK  460 trk=asc(tl$(x)): sec=asc(sl$(x))
LE  470 next x: close5: close15: end
```

Machine Language Print Loader

Noel Nyman
Portland, Oregon

PRINT. . .NOT POKE. . .Machine Language Programs

Editor's note: The BASIC loader program accompanying this article is quite long, but the article contains concepts which can be digested independently of the program. If you want a copy of the accompanying program but don't feel like typing it in, you can find it on the Transactor disk for this issue.

BASIC programs often use machine language to read disks, sort data, or accomplish something more quickly and easily than can be done with BASIC alone. The common ways to use ML with BASIC are: LOAD the ML from disk or tape from the BASIC program, append the ML to BASIC, or use READ/DATA/POKE statements.

Each of these has its place. But in this article we'll explore another method. . .using BASIC PRINT statements to "poke" ML programs.

First, type in the following line exactly as shown. If a letter is capitalized, hold the SHIFT key when typing it. For commands such as (com–m) hold the Commodore key and type"m." Press the SPACE bar when you see (SPACE).

```
20 ? "(rvs–on)(SPACE)(rvs–off)@(rvs–on)9(rvs–off)
   (com–m) [ (rvs–on)(com–a)(rvs–off)g(SPACE)(rvs–on)R(com–b)
   H(rvs–off)LZ [ (Shf–(SPACE))(rvs–on)s(rvs–off)THIS(SPACE)
   IS(SPACE)THE(SPACE)ML(SPACE)DATA@"
```

Now type RUN. You should see thirty–seven characters printed to the screen. Shifting to lower case mode may help you count them.

Next type this additional line:

```
10 POKE209,88:POKE210,27:POKE211,0:POKE213,40
```

Type RUN again. This time you should see only the READY prompt. Type SYS 7000, the screen will clear and the ML message will be displayed.

We normally think of the PRINT command as "printing" characters on the screen starting from the current cursor location. Of course, we know that PRINT just stores a sequence of data into computer memory. The VIC chip reads that data and creates the screen we see.

This may seem like a trivial distinction at first. But we can use the sequential storing characteristic of PRINT to "poke" ML programs and data. All we have to do is move the "cursor" away from screen memory to the area where we want the ML to reside, then put the ML data into PRINT statements.

The cursor position is controlled by several memory locations. We'll be concerned with four of them. Locations 209 and 210 together form the cursor current line "vector", the address of the beginning of the line the cursor is on. The vector is stored in usual 6502 format, low byte first.

Location 211 identifies cursor location on the line by column number. Location 213 tells the system how long a line is. The usual values for 213 are 39 or 79 for a C–64, and 21, 43, 65, or 87 for a VIC.

These numbers need to be in the proper ranges, or your screen will look strange. But when we use PRINT to store ML, we can change them to our advantage. In line #10 above, we re–vectored the cursor to start at column zero on a "line" beginning at address 7000. We then told the computer that this line is forty columns long. We could have used any number up to 255 for line length, so long as it was greater than the number of bytes we wanted to store.

After you typed RUN, you saw the READY prompt. This shows you that the cursor will be vectored back to the normal screen memory by the operating system at the end of each PRINT statement. Actually, we could use several PRINT statements ending with ';' and the data will be stored continuously, just as it would be shown continuously on the screen. If we want to store over 255 bytes of ML, we'll need another POKE line with new values at locations 209 and 210.

PRINT puts "screen codes" in computer memory, not CHR$ values. The USER'S GUIDE that came with your machine has a table of screen codes in an appendix. If you compare them with the CHR$ codes, you'll notice that there are only 128 screen codes. Also they are often in the same sequence as the CHR$ codes, but start at different places. For example, the alphabet is in order. However, CHR$(65) which is the letter 'A' has a screen code of 1. Therefore, to get a 1 stored in memory we PRINT an 'A'. To store a 65, we have to PRINT a 'spade' symbol.

Another problem shows up if we need to store '34'. This is the screen code for quote marks. If the PRINT command sees quote marks, it will end string printing and try to cope with the balance of the characters on the line as variables. A special technique is required to PRINT a "34" into memory.

Since it's so much work to create the PRINT statements, are there any advantages over the other methods? There are, especially if you use a program to create the PRINT statements.

Of the methods mentioned at the start of this article, READ/DATA/POKE is the most common. For short ML it saves time over LOADing the ML routine separately. It also avoids having to store the ML as a separate program on the same disk or tape. It has two disadvantages, however. It's slow and it takes a big chunk of memory. Memory saving is most important to VIC owners. But anyone could take advantage of improved speed.

Storing ML at the end of a BASIC program is becoming more common. It requires a few POKEs to convince the system to SAVE the ML along with BASIC, and even more cleverness if the BASIC program is changed. It also reduces available variable memory to VIC owners, since the ML sits in BASIC memory area.

By using PRINT statements to store your ML programs, you can save BASIC memory and improve program run time over the READ/DATA/POKE method. PRINT works eight to thirteen times faster than READ/POKE. The exact memory saving depends on the ML program. But it usually takes less than half the BASIC memory to store ML in PRINT statements over DATA statements.

The PRINT lines are a part of BASIC, not an ML appendage. So any editing done to the BASIC program will not affect them.

Program Operation

To make using this technique easier, we've provided ML "PRINTER" programs for the VIC and the C-64 that will create the PRINT and POKE statements in BASIC. We'll look at the C-64 program first. Listing #1 is a READ/DATA/POKE program to put ML "PRINTER" into C-64 memory. Type in the program, SAVE it, then RUN it. Then type SYS 25856.

The screen will prompt you for the beginning and ending locations of the ML you want to "copy", the number of the first BASIC line to be created, and the increment for the following BASIC lines. You'll be given the option of entering the ML locations in hex or decimal. Type in any locations to test the program, 828 and 1019 (the cassette buffer) for example. After the READY prompt, type LIST. You'll see the newly created BASIC, which has been written over the READ/DATA/POKE program.

You could SAVE these lines to be appended to a BASIC program or continue entering a BASIC program using them.

To use ML "PRINTER" most effectively, you should have it as a ML program. (You can use it to make a BASIC PRINT version of itself, but an ML version is shorter and for a program this size, more efficient.) If you have a monitor, you can use it to save the program. If not, type the following exactly as shown:

```
CLR (RETURN)
POKE43,0 :POKE44,101 :POKE45,185 :POKE46,108 (RETURN)
```

This moves the start and end of BASIC to enclose ML "PRINTER." Now SAVE the program to tape or disk just as you would any BASIC program. After the SAVE is complete use SYS 64738 to reset the computer. You can now LOAD ML "PRINTER" using a ',1' after the LOAD statement.

This program will work best for you if you use it on relatively short ML routines. If you go above about 1.5K bytes, it's faster to LOAD ML routines from disk. For example, using ML "PRINTER" to store DOS 5.1 as a BASIC "PRINT" program requires loading only one program. But it takes about three seconds longer to PRINT than it does to RUN the boot program which LOADs the normal ML version from the disk.

If you use ML "PRINTER" to PRINT over 1K of ML, 1025 bytes or more, you'll find that BASIC will work fine. But your keyboard won't. Apparently when the screen gets longer than 1024 bytes, the operating system is confused and the CIA chips get "lost." If you SYS to the ML immediately after printing you should have no problems. If not, follow the last PRINT statement with a BASIC line that contains:

```
SYS 64931:PRINT "(shift/clr-home)"
```

This will put everything right again. The "clear-screen" command is a good idea in any case to restore the normal line length parameters. Since you usually need to get the ML in place early in the BASIC program, you could put the PRINT statements (or a GOSUB to them) ahead of a "clear-screen" already in the program.

The C-64 program has been located in an area unlikely to have ML residing in it. The program may give strange results if the interrupt system has been changed by a previous program, such as the DOS wedge. It's best to use a full RESTORE, such as SYS 64738, before loading and using ML "PRINTER." You may need to use the NEW command after loading ML "PRINTER" to load the ML you want to "copy."

The VIC version was written for the unexpanded VIC-20, primarily to "copy" ML in the cassette buffer. Expanded VIC owners will want to use a variation on the C-64 program, to be discussed later.

There are two READ/DATA/POKE programs for the VIC, because there isn't enough room in VIC memory to hold all the DATA. A PRINT statement version will fit in VIC memory easily, however, which shows how much more compact it is.

Type and SAVE each program. To save on memory, we've left the screen prompts out of VIC ML "PRINTER." Program #4, VIC SCREEN, is a BASIC program that will get the various numbers the "PRINTER" needs. Type in and SAVE this program also.

LOAD and RUN each of the VIC DATA programs in sequence. This will POKE the ML portion of the "PRINTER" in place. Then type the following directly to the screen:

```
POKE55,0 :POKE56,25 (RETURN)
NEW:CLR (RETURN)
```

This will move the end of BASIC memory below ML "PRINTER" to protect it. Now LOAD the VIC SCREEN program and RUN it. It will ask for the same information as the C-64 version, except that the ML locations must be entered in decimal (base 10). To SAVE VIC ML "PRINTER" we'll use it to make a PRINT statement version of itself.

Type 6927 and 7678 to answer the prompts for starting and ending ML locations. Use ten as a starting line for BASIC, and one as the increment.

When you see the READY prompt, type SYS 6927 and hit RETURN.

Then type LIST. VIC SCREEN has been replaced with POKE and PRINT statements that will create ML "PRINTER." To make this version complete, you'll want to append VIC SCREEN to it. To do this, you move the start of BASIC to the end of the present BASIC program by typing the following directly:

```
CLR (RETURN)
POKE43,PEEK(45)-2:POKE46,PEEK(44) (RETURN)
```

Now LOAD VIC SCREEN. When READY appears, reset the BASIC pointers by typing:

```
POKE43,1:POKE44,16 (RETURN)
```

LIST and you'll see VIC SCREEN added onto the POKE/PRINT lines. Then add the following line:

1150 SYS 6927 (RETURN)

SAVE this completed version of VIC ML "PRINTER."

Both versions of ML "PRINTER" update the BASIC start and end parameters after the new BASIC lines are created. Since the BASIC memory location is different for different Commodore computers, the VIC and C-64 versions use different values. Expanded VICs have BASIC relocated, so the VIC ML "PRINTER" won't work with them. If you have an expanded VIC that has memory in the range 25856 to 27500, you can use the C-64 "PRINTER" with a few changes. These can be made easily with any VIC monitor programs such as VICMON.

The data from $6500 to $6508 are C-64 border and screen color commands. Change them to $EA for NOP. Change the starting address information in locations $666B and $6667 to correspond to your beginning BASIC memory area, PEEK(43) and PEEK(44). Subtract one from PEEK(43) and put the result in $6671. Change $$6672 to the PEEK(44) value. Change the data at $676C and $6776 to correspond to your normal top of BASIC, PEEK(55) and PEEK(56). The program should then function correctly, although the screen will still be in C-64 format.

To append the POKE/PRINT statements to another BASIC program, you can use the following technique with the C-64 or any VIC. LOAD the program with the lowest line numbers. The lowest line number in the program to be appended must be higher than the highest line number of the first program. ML "PRINTER" gives you the option of selecting line numbers to make this easy. Type:

PRINT PEEK(43),PEEK(44) (press RETURN)

Write down the two numbers that appear on the screen as "x" and "y." This is the start of BASIC memory in your computer. Type:

PRINT PEEK(45) (RETURN)

This is the lower byte of the address of the end of the BASIC program you just LOADed. If this number is not zero or one, type:

POKE43,PEEK(45)-2:POKE44,PEEK(46) (RETURN)

If PEEK(45) gave you zero or one, type:

POKE43,PEEK(45)+254:POKE44,PEEK(46)-1 (RETURN)

These statements move the start of BASIC memory to the end of the program. Now LOAD the second program as a BASIC program. Then type:

POKE43,x:POKE44,y (RETURN)

Where "x" and "y" are the two numbers your wrote down earlier. This moves the start of BASIC back to the beginning of the first program. LIST will now show the second program appended to the first.

I'd like to thank Barbara Horton for her help in developing the VIC version of this program.

Listing 1: BASIC loader for C64 version

```
FL 10 rem* data loader for " print ml " *
LI 20 cs=0
OF 30 for i=25856 to 27832:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
DC 60 if cs<>203777 thenprint " **** error in data
      statments **** " : end
IJ 70 rem sys 25856
AF 80 end
IN 100 :
LG 1000 data 160,  6, 140,  32, 208, 200
OD 1010 data 140,  33, 208, 169, 213, 133
FA 1020 data 139, 169, 106, 133, 140,  32
ED 1030 data 137, 103, 169, 108, 133, 140
FD 1040 data 169,  96, 133, 139, 169,   1
AK 1050 data 141, 167,   2,  32, 151, 103
DI 1060 data 173, 192,   2,  56, 233,  72
FI 1070 data 240,   8, 169,   1, 141, 254
CH 1080 data   2,  76,  55, 101, 141, 254
OL 1090 data   2,  32,   6, 104, 169,  85
HI 1100 data 133, 139, 169, 107, 133, 140
ID 1110 data  32, 137, 103, 173, 254,   2
IG 1120 data  24, 105,   4, 141, 167,   2
OI 1130 data 201,   5, 240,   7, 169, 114
EN 1140 data 133, 139,  76,  93, 101, 169
LK 1150 data 101, 133, 139, 169, 108, 133
DL 1160 data 140,  32, 151, 103, 173, 254
LA 1170 data   2, 201,   1, 208,   6,  32
CD 1180 data  24, 104,  76, 116, 101,  32
CN 1190 data 114, 104, 173, 171,   2, 141
LN 1200 data 252,   2, 173, 172,   2, 141
AP 1210 data 253,   2,  32,   6, 104, 169
DP 1220 data 110, 133, 139, 169, 107, 133
LA 1230 data 140,  32, 137, 103, 173, 254
GF 1240 data   2, 208,  22, 169, 114, 133
AD 1250 data 139, 169, 108, 133, 140, 169
IO 1260 data   4, 141, 167,   2,  32, 151
PE 1270 data 103,  32, 114, 104,  76, 188
ID 1280 data 101, 169, 101, 133, 139, 169
PD 1290 data 108, 133, 140, 169,   5, 141
AD 1300 data 167,   2,  32, 151, 103,  32
MN 1310 data  24, 104, 173, 171,   2, 141
BF 1320 data 250,   2, 173, 172,   2, 141
GG 1330 data 251,   2,  32,   6, 104, 169
GH 1340 data 133, 133, 139, 169, 107, 133
LH 1350 data 140,  32, 137, 103, 169, 101
MI 1360 data 133, 139, 169, 108, 133, 140
EG 1370 data 169,   5, 141, 167,   2,  32
DL 1380 data 151, 103, 173, 192,   2, 208
LO 1390 data  39, 169,  49, 141, 192,   2
MF 1400 data  32, 210, 255, 169,  48,  32
MC 1410 data 210, 255, 141, 193,   2,  32
HD 1420 data 210, 255, 141, 194,   2,  32
CE 1430 data 210, 255, 141, 195,   2,  32
OP 1440 data 210, 255, 141, 196,   2, 169
HO 1450 data   5, 141, 168,   2,  32,  24
PO 1460 data 104, 173, 171,   2, 141, 248
MO 1470 data   2, 173, 172,   2, 141, 249
KB 1480 data   2,  32,   6, 104, 169, 196
NA 1490 data 133, 139, 169, 107, 133, 140
JB 1500 data  32, 137, 103, 169, 101, 133
BP 1510 data 139, 169, 108, 133, 140,  32
PD 1520 data 151, 103, 173, 192,   2, 208
EF 1530 data  21, 169,  49, 141, 192,   2
NH 1540 data  32, 210, 255, 169,  48, 141
```

```
LK 1550 data 193,   2,  32, 210, 255, 169
ME 1560 data   2, 141, 168,   2,  32,  24
HF 1570 data 104, 173, 171,   2, 141, 246
IF 1580 data   2, 173, 172,   2, 141, 247
DB 1590 data   2,  32,   6, 104, 169,   8
OF 1600 data 133, 114, 169,   1, 133, 113
EH 1610 data 169,   0, 141,   0,   8, 141
FH 1620 data 244,   2, 141, 245,   2, 141
NI 1630 data 237,   2, 141, 236,   2, 173
FJ 1640 data 253,   2, 133, 248, 173, 252
MJ 1650 data   2, 133, 247, 173, 249,   2
BL 1660 data 141, 243,   2, 173, 248,   2
ML 1670 data 141, 242,   2, 169, 107, 133
HK 1680 data 140, 169, 241, 133, 139,  32
KO 1690 data 137, 103,  32, 174, 105,  32
HO 1700 data 239, 105,  32,  62, 106, 169
IO 1710 data   0, 141, 237,   2, 168, 177
CA 1720 data 247, 133, 254, 201, 128, 176
HN 1730 data  21, 173, 245,   2, 201,   1
FD 1740 data 208,  32, 169, 146,  32, 232
OP 1750 data 104, 206, 245,   2, 238, 244
MF 1760 data   2,  76, 222, 102, 173, 245
NO 1770 data   2, 201,   1, 240,  11, 169
JC 1780 data  18,  32, 232, 104, 238, 245
AE 1790 data   2, 238, 244,   2, 165, 254
PP 1800 data  10,  74, 201,  34, 208,   6
OG 1810 data  76,  79, 106,  76,  24, 103
BL 1820 data 133, 254,  10,  10, 176,  16
GH 1830 data  10, 144,   5, 165, 254,  76
CK 1840 data  18, 103, 165, 254,  24, 105
EI 1850 data  64,  76,  18, 103,  10, 144
CI 1860 data   8, 165, 254,  24, 105,  64
PP 1870 data  76,  18, 103, 165, 254,  24
CH 1880 data 105, 128,  32, 232, 104, 238
GJ 1890 data 244,   2, 238, 236,   2, 230
FJ 1900 data 247, 208,   2, 230, 248, 173
CL 1910 data 250,   2, 197, 247, 173, 251
PI 1920 data   2, 229, 248, 176,   6,  32
PM 1930 data 143, 106,  76,  91, 103, 173
AL 1940 data 236,   2, 201, 255, 208,   6
NP 1950 data  32, 172, 106,  76, 149, 102
MD 1960 data 173, 244,   2, 201,  57, 144
ON 1970 data  11,  32, 194, 106,  32,  62
DN 1980 data 106, 169,   0, 141, 244,   2
JB 1990 data  76, 169, 102, 230, 113, 208
LM 2000 data   2, 230, 114, 165, 113, 133
KJ 2010 data  45, 133,  47, 133,  49, 165
KD 2020 data 114, 133,  46, 133,  48, 133
HE 2030 data  50, 169,   0, 133,  51, 133
NK 2040 data  53, 133,  54, 133,  55, 169
LG 2050 data 128, 133,  56, 133,  52, 169
GB 2060 data   1, 133,  43, 169,   8, 133
JI 2070 data  44, 169, 147,  32, 210, 255
PN 2080 data  96, 160,   0, 177, 139, 240
GF 2090 data   7,  32, 210, 255, 200,  76
JA 2100 data 139, 103,  96, 169,   0, 141
CH 2110 data 168,   2, 169, 192, 133, 141
MF 2120 data 169,   2, 133, 142, 160,   0
LA 2130 data 177, 139, 141, 169,   2,  32
GL 2140 data 159, 255,  32, 228, 255, 172
NJ 2150 data 169,   2, 209, 139, 240,   6
LJ 2160 data 136, 240, 233,  76, 180, 103
PO 2170 data 201,  20, 240,  54, 201,  13
EO 2180 data 208,  15, 168, 165, 197, 201
JF 2190 data  64, 208, 250, 162,   0, 202
GN 2200 data 138, 208, 252, 152,  96, 172
EN 2210 data 168,   2, 170, 173, 167,   2
BO 2220 data 205, 168,   2, 240, 195, 138
IP 2230 data  32, 210, 255, 145, 141, 238
OF 2240 data 168,   2, 165, 197, 201,  64
GA 2250 data 240, 250, 160,   0, 136, 208
BE 2260 data 253,  76, 164, 103, 172, 168
BA 2270 data   2, 240, 167, 206, 168,   2
PC 2280 data  32, 210, 255,  76, 234, 103
HL 2290 data 160,  15, 169,   2, 133, 142
BG 2300 data 169, 192, 133, 141, 169,   0
DH 2310 data 145, 141, 136,  16, 251,  96
IF 2320 data 173, 168,   2, 170, 169,   0
JD 2330 data 141, 170,   2, 141, 171,   2
GG 2340 data 141, 172,   2, 169, 192, 133
GF 2350 data 141, 169,   2, 133, 142, 172
CG 2360 data 170,   2, 177, 141,  72,  14
EF 2370 data 171,   2,  46, 172,   2, 173
HE 2380 data 171,   2, 172, 172,   2,  14
DJ 2390 data 171,   2,  46, 172,   2,  14
PJ 2400 data 171,   2,  46, 172,   2,  24
CI 2410 data 109, 171,   2, 141, 171,   2
PJ 2420 data 152, 109, 172,   2, 141, 172
HH 2430 data   2, 104,  56, 233,  48,  24
AK 2440 data 109, 171,   2, 141, 171,   2
JM 2450 data 144,   3, 238, 172,   2, 238
JC 2460 data 170,   2, 202, 208, 190,  96
NF 2470 data 173, 195,   2, 208,  26, 173
JP 2480 data 194,   2, 141, 195,   2, 173
PP 2490 data 193,   2, 141, 194,   2, 173
HA 2500 data 192,   2, 141, 193,   2, 169
IE 2510 data  48, 141, 192,   2,  76, 114
GO 2520 data 104, 169,   0, 141, 170,   2
NE 2530 data 169, 192, 133, 141, 169,   2
MB 2540 data 133, 142, 160,   0, 177, 141
KC 2550 data 141, 174,   2, 238, 170,   2
BF 2560 data 172, 170,   2, 177, 141, 168
JE 2570 data 173, 174,   2,  72, 152,  32
KD 2580 data 222, 104, 141, 173,   2, 104
ID 2590 data  32, 222, 104,  10,  10,  10
HG 2600 data  10,  13, 173,   2, 168, 173
CC 2610 data 170,   2, 201,   3, 240,  13
NH 2620 data 152, 141, 172,   2, 238, 170
NE 2630 data   2, 172, 170,   2,  76, 160
FP 2640 data 104, 152, 141, 171,   2,  96
OA 2650 data  56, 233,  48, 201,  16, 144
HC 2660 data   2, 233,   7,  96, 162,   0
KI 2670 data 129, 113, 230, 113, 208,   2
KP 2680 data 230, 114,  96, 169,  48, 141
PL 2690 data 165, 108, 141, 166, 108, 141
OO 2700 data 167, 108, 166, 248, 232, 169
II 2710 data   0, 202, 240,  43,  24, 105
LL 2720 data   1, 201,  10, 240,   3,  76
CM 2730 data   3, 105, 169,   1,  24, 109
IH 2740 data 166, 108, 201,  58, 240,   6
JH 2750 data 141, 166, 108,  76,   1, 105
OE 2760 data 169,  48, 141, 166, 108, 169
NE 2770 data   1,  24, 109, 165, 108, 141
KC 2780 data 165, 108,  76,   1, 105,  24
AI 2790 data 105,  48, 141, 167, 108,  96
NE 2800 data 169,  48, 141, 156, 108, 141
AE 2810 data 157, 108, 141, 158, 108, 166
JC 2820 data 247, 232, 169,   0, 202, 240
DP 2830 data  43,  24, 105,   1, 201,  10
GG 2840 data 240,   3,  76,  72, 105, 169
AK 2850 data   1,  24, 109, 157, 108, 201
GO 2860 data  58, 240,   6, 141, 157, 108
```

```
NO 2870 data  76,  70, 105, 169,  48, 141
CO 2880 data 157, 108, 169,   1,  24, 109
JG 2890 data 156, 108, 141, 156, 108,  76
PO 2900 data  70, 105,  24, 105,  48, 141
AK 2910 data 158, 108,  96, 165, 113, 141
GH 2920 data 240,   2, 165, 114, 141, 241
GI 2930 data   2,  32, 232, 104,  32, 232
JO 2940 data 104,  96, 173, 242,   2,  32
PB 2950 data 232, 104, 173, 243,   2,  32
KF 2960 data 232, 104,  24, 173, 246,   2
IK 2970 data 109, 242,   2, 141, 242,   2
AO 2980 data 173, 247,   2, 109, 243,   2
PO 2990 data 141, 243,   2,  96,  32, 243
BE 3000 data 104,  32,  56, 105,  32, 125
JG 3010 data 105,  32, 142, 105, 160,   0
KB 3020 data 169, 108, 133, 140, 169, 151
HD 3030 data 133, 139, 177, 139, 240,   7
HD 3040 data  32, 232, 104, 200,  76, 196
GG 3050 data 105, 169,   0,  32, 232, 104
KE 3060 data  32, 216, 105,  96, 173, 240
NP 3070 data   2, 133, 141, 173, 241,   2
CD 3080 data 133, 142, 162,   0, 165, 113
JE 3090 data 129, 141, 230, 141, 165, 114
LP 3100 data 129, 141,  96, 169,   0, 141
FE 3110 data 245,   2, 141, 244,   2, 141
HF 3120 data 237,   2, 169, 108, 133, 140
BN 3130 data 169,  59, 133, 139,  32, 137
EJ 3140 data 103, 165, 248,  32,  38, 106
DN 3150 data  32, 210, 255, 165, 248,  32
GJ 3160 data  54, 106,  32, 210, 255, 165
PP 3170 data 247,  32,  38, 106,  32, 210
MM 3180 data 255, 165, 247,  32,  54, 106
GO 3190 data  32, 210, 255,  96,  74,  74
IM 3200 data  74,  74,  24, 105,  48, 201
GP 3210 data  58, 176,   1,  96,  24, 105
GB 3220 data   7,  96,  10,  10,  10,  10
BO 3230 data  32,  38, 106,  96,  32, 125
AO 3240 data 105,  32, 142, 105, 169, 153
BI 3250 data  32, 232, 104, 169,  34,  32
KA 3260 data 232, 104,  96,  32, 194, 106
DF 3270 data  32,  62, 106, 165, 254,  10
AK 3280 data 144,  15, 169,  18,  32, 232
HA 3290 data 104, 169,  34,  32, 232, 104
NC 3300 data 169,  34,  32, 232, 104, 198
JA 3310 data 113, 208,   2, 198, 114, 160
DC 3320 data   1, 169, 108, 133, 140, 169
AF 3330 data 133, 133, 139, 177, 139, 240
KA 3340 data  10,  32, 232, 104, 200, 238
HK 3350 data 244,   2,  76, 121, 106, 169
FG 3360 data   1, 141, 237,   2,  76,  24
OF 3370 data 103, 173, 237,   2, 208,   5
KH 3380 data 169,  34,  32, 232, 104, 169
IE 3390 data   0,  32, 232, 104,  32, 216
EM 3400 data 105, 169,   0,  32, 232, 104
MI 3410 data 169,   0,  32, 232, 104,  96
OK 3420 data 173, 237,   2, 208,   5, 169
AJ 3430 data  34,  32, 232, 104, 169,   0
CP 3440 data 141, 236,   2,  32, 232, 104
BF 3450 data  32, 216, 105,  96, 169,  34
GG 3460 data  32, 232, 104, 169,  59,  32
FB 3470 data 232, 104, 169,   0,  32, 232
JP 3480 data 104,  32, 216, 105,  96, 147
CJ 3490 data  32,  32, 142,  32,  32,  32
GO 3500 data  32, 151,  18,  32,  77,  65
FF 3510 data  67,  72,  73,  78,  69,  32
NG 3520 data  76,  65,  78,  71,  85,  65
JD 3530 data  71,  69,  32,  34,  80,  82
IG 3540 data  73,  78,  84,  69,  82,  34
GB 3550 data  32,  13,  13,  32,  32,  32
PC 3560 data  32,  32,  32,  32,  32,  18
OE 3570 data  32,  40,  67,  41,  32,  49
JJ 3580 data  57,  56,  52,  32,  66,  89
KJ 3590 data  32,  78,  79,  69,  76,  32
NM 3600 data  78,  89,  77,  65,  78,  32
NH 3610 data  13,  13,  13,  31,  87,  73
DO 3620 data  76,  76,  32,  89,  79,  85
HN 3630 data  32,  69,  78,  84,  69,  82
NM 3640 data  32,  76,  79,  67,  65,  84
LM 3650 data  73,  79,  78,  83,  13,  32
AA 3660 data  73,  78,  32,  72,  69,  88
NO 3670 data  32,  79,  82,  32,  68,  69
KP 3680 data  67,  73,  77,  65,  76,  32
LM 3690 data  40,  72,  47,  68,  41,  63
AC 3700 data   0,  13,  13,  83,  84,  65
KP 3710 data  82,  84,  73,  78,  71,  32
GE 3720 data  77,  76,  32,  76,  79,  67
KD 3730 data  65,  84,  73,  79,  78,  63
LF 3740 data  32,   0,  13,  13,  69,  78
BE 3750 data  68,  73,  78,  71,  32,  77
EF 3760 data  76,  32,  76,  79,  67,  65
KE 3770 data  84,  73,  79,  78,  63,  32
LH 3780 data   0,  13,  13,  76,  73,  78
FI 3790 data  69,  32,  78,  85,  77,  66
AI 3800 data  69,  82,  32,  70,  79,  82
KF 3810 data  32,  70,  73,  82,  83,  84
DH 3820 data  32,  80,  82,  73,  78,  84
GI 3830 data  32,  83,  84,  65,  84,  69
EJ 3840 data  77,  69,  78,  84,  13,  40
LK 3850 data  18,  82,  69,  84,  85,  82
BH 3860 data  78, 146,  32,  70,  79,  82
EL 3870 data  32,  49,  48,  48,  48,  48
IL 3880 data  41,  63,  32,   0,  13,  13
PM 3890 data  76,  73,  78,  69,  32,  78
EN 3900 data  85,  77,  66,  69,  82,  32
GA 3910 data  73,  78,  67,  82,  69,  77
LM 3920 data  69,  78,  84,  32,  13,  40
LP 3930 data  18,  82,  69,  84,  85,  82
BM 3940 data  78, 146,  32,  70,  79,  82
LN 3950 data  32,  49,  48,  41,  63,  32
EM 3960 data   0, 147,  17,  17,  17,  17
KP 3970 data  17,  17,  17,  17,  17,  29
NC 3980 data  29,  29,  29,  29,  29,  29
KD 3990 data  29,  67,  82,  69,  65,  84
GB 4000 data  73,  78,  71,  32,  80,  82
JD 4010 data  73,  78,  84,  32,  83,  84
HH 4020 data  65,  84,  69,  77,  69,  78
MD 4030 data  84,  13,  13,  29,  29,  29
JG 4040 data  29,  29,  29,  29,  29,  29
KI 4050 data  29,  29,  70,  82,  79,  77
PG 4060 data  32,  65,  68,  68,  82,  69
PD 4070 data  83,  83,  32,  32,  32,  32
AL 4080 data  32,  32,   0,  19,  17,  17
KG 4090 data  17,  17,  17,  17,  17,  17
NI 4100 data  17,  17,  17,  29,  29,  29
PK 4110 data  29,  29,  29,  29,  29,  29
JL 4120 data  29,  29,  29,  29,  29,  29
DM 4130 data  29,  29,  29,  29,  29,  29
KJ 4140 data  29,  29,  32,   0,   4,  68
PJ 4150 data  72,  13,  20,  12,  48,  49
OJ 4160 data  50,  51,  52,  53,  54,  55
MM 4170 data  56,  57,  13,  20,  18,  48
GM 4180 data  49,  50,  51,  52,  53,  54
```

```
CA 4190 data  55,  56,  57,  65,  66,  67
BN 4200 data  68,  69,  70,  13,  20,  34
LN 4210 data 199,  40,  51,  52,  41, 199
BJ 4220 data  40,  51,  52,  41, 199,  40
HL 4230 data  50,  48,  41,  34,   0, 151
CB 4240 data  50,  48,  57,  44,  48,  54
ON 4250 data  48,  58, 151,  50,  49,  48
HP 4260 data  44,  48,  48,  51,  58, 151
CE 4270 data  50,  49,  49,  44,  48,  58
CN 4280 data 151,  50,  49,  51,  44,  50
KJ 4290 data  53,  53,   0,
```

Listing 2: first loader program for VIC–20 version

```
KK 10 rem* first data loader for "printing ml" *
LI 20 cs=0
LL 30 for i=6927 to 7302:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
JI 60 if cs<>43462thenprint"**** error in data
      statments ****": end
AF 80 end
IN 100 :
IG 1000 data 160,  16, 132, 114, 160,   1, 132, 113
HK 1010 data 136, 140,   0,  16, 140, 244,   2, 140
MP 1020 data 245,   2, 140, 237,   2, 140, 236,   2
PA 1030 data 173, 253,   2, 133, 248, 173, 252,   2
NB 1040 data 133, 247, 173, 249,   2, 141, 243,   2
KO 1050 data 173, 248,   2, 141, 242,   2,  32, 231
KL 1060 data  28,  32,  40,  29,  32,  52,  29, 169
HE 1070 data   0, 141, 237,   2, 168, 177, 247, 133
EO 1080 data 254, 201, 128, 176,  21, 173, 245,   2
CA 1090 data 201,   1, 208,  32, 169, 146,  32,  33
PK 1100 data  28, 206, 245,   2, 238, 244,   2,  76
FF 1110 data 123,  27, 173, 245,   2, 201,   1, 240
IK 1120 data  11, 169,  18,  32,  33,  28, 238, 245
NI 1130 data   2, 238, 244,   2, 165, 254,  10,  74
MF 1140 data 201,  34, 208,   6,  76,  69,  29,  76
MK 1150 data 181,  27, 133, 254,  10,  10, 176,  16
OJ 1160 data  10, 144,   5, 165, 254,  76, 175,  27
BN 1170 data 165, 254,  24, 105,  64,  76, 175,  27
GI 1180 data  10, 144,   8, 165, 254,  24, 105,  64
PE 1190 data  76, 175,  27, 165, 254,  24, 105, 128
BM 1200 data  32,  33,  28, 238, 244,   2, 238, 236
DM 1210 data   2, 230, 247, 208,   2, 230, 248, 173
BN 1220 data 250,   2, 197, 247, 173, 251,   2, 229
HP 1230 data 248, 176,   6,  32, 133,  29,  76, 248
BP 1240 data  27, 173, 236,   2, 201, 255, 208,   6
BC 1250 data  32, 162,  29,  76,  61,  27, 173, 244
HA 1260 data   2, 201,  57, 144,  11,  32, 184,  29
LI 1270 data  32,  52,  29, 169,   0, 141, 244,   2
MN 1280 data  76,  70,  27, 230, 113, 208,   2, 230
DD 1290 data 114, 165, 113, 133,  45, 133,  47, 133
NM 1300 data  49, 165, 114, 133,  46, 133,  48, 133
CM 1310 data  50, 169, 255, 133,  51, 133,  53, 133
KG 1320 data  54, 133,  55, 169,  29, 133,  56, 133
LF 1330 data  52, 169,   1, 133,  43, 169,  16, 133
LE 1340 data  44,  96, 162,   0, 129, 113, 230, 113
MD 1350 data 208,   2, 230, 114,  96, 169,  48, 141
GA 1360 data 235,  29, 141, 236,  29, 141, 237,  29
DM 1370 data 166, 248, 232, 169,   0, 202, 240,  43
IN 1380 data  24, 105,   1, 201,  10, 240,   3,  76
DO 1390 data  60,  28, 169,   1,  24, 109, 236,  29
DI 1400 data 201,  58, 240,   6, 141, 236,  29,  76
FN 1410 data  58,  28, 169,  48, 141, 236,  29, 169
EH 1420 data   1,  24, 109, 235,  29, 141, 235,  29
AN 1430 data  76,  58,  28,  24, 105,  48, 141, 237
FO 1440 data  29,  96, 169,  48, 141, 226,  29, 141
CA 1450 data 227,  29, 141, 228,  29, 166, 247, 232
NI 1460 data 169,   0, 202, 240,  43,  24, 105,   1
```

Listing 3: second loader for VIC version

```
KB 10 rem* second data loader for "printing ml" *
JJ 15 rem* vic version *
LI 20 cs=0
FM 30 for i=7303 to 7678:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
CJ 60 if cs<>33737thenprint"**** error in data
      statements ****": end
AF 80 end
IN 100 :
CA 1000 data 201,  10, 240,   3,  76, 129,  28, 169
CN 1010 data   1,  24, 109, 227,  29, 201,  58, 240
BC 1020 data   6, 141, 227,  29,  76, 127,  28, 169
ME 1030 data  48, 141, 227,  29, 169,   1,  24, 109
CG 1040 data 226,  29, 141, 226,  29,  76, 127,  28
IG 1050 data  24, 105,  48, 141, 228,  29,  96, 165
AA 1060 data 113, 141, 240,   2, 165, 114, 141, 241
CD 1070 data   2,  32,  33,  28,  32,  33,  28,  96
NE 1080 data 173, 242,   2,  32,  33,  28, 173, 243
IF 1090 data   2,  32,  33,  28,  24, 173, 246,   2
JE 1100 data 109, 242,   2, 141, 242,   2, 173, 247
DM 1110 data   2, 109, 243,   2, 141, 243,   2,  96
KI 1120 data  32,  44,  28,  32, 113,  28,  32, 182
PM 1130 data  28,  32, 199,  28, 160,   0, 169,  29
CK 1140 data 133, 140, 169, 221, 133, 139, 177, 139
AL 1150 data 240,   7,  32,  33,  28, 200,  76, 253
BO 1160 data  28, 169,   0,  32,  33,  28,  32,  17
GM 1170 data  29,  96, 173, 240,   2, 133, 141, 173
DI 1180 data 241,   2, 133, 142, 162,   0, 165, 113
FL 1190 data 129, 141, 230, 141, 165, 114, 129, 141
HN 1200 data  96, 169,   0, 141, 245,   2, 141, 244
ND 1210 data   2, 141, 237,   2,  96,  32, 182,  28
EP 1220 data  32, 199,  28, 169, 153,  32,  33,  28
OB 1230 data 169,  34,  32,  33,  28,  96,  32, 184
BO 1240 data  29,  32,  52,  29, 165, 254,  10, 144
DH 1250 data  15, 169,  18,  32,  33,  28, 169,  34
KH 1260 data  32,  33,  28, 169,  34,  32,  33,  28
JP 1270 data 198, 113, 208,   2, 198, 114, 160,   1
HF 1280 data 169,  29, 133, 140, 169, 203, 133, 139
CB 1290 data 177, 139, 240,  10,  32,  33,  28, 200
OC 1300 data 238, 244,   2,  76, 111,  29, 169,   1
KD 1310 data 141, 237,   2,  76, 181,  27, 173, 237
FE 1320 data   2, 208,   5, 169,  34,  32,  33,  28
IF 1330 data 169,   0,  32,  33,  28,  32,  17,  29
II 1340 data 169,   0,  32,  33,  28, 169,   0,  32
CP 1350 data  33,  28,  96, 173, 237,   2, 208,   5
AH 1360 data 169,  34,  32,  33,  28, 169,   0, 141
KH 1370 data 236,   2,  32,  33,  28,  32,  17,  29
MB 1380 data  96, 169,  34,  32,  33,  28, 169,  59
LH 1390 data  32,  33,  28, 169,   0,  32,  33,  28
NE 1400 data  32,  17,  29,  96,  34, 199,  40,  51
FM 1410 data  52,  41, 199,  40,  51,  52,  41, 199
BK 1420 data  40,  50,  48,  41,  34,   0, 151,  50
EE 1430 data  48,  57,  44,  50,  53,  50,  58, 151
CP 1440 data  50,  49,  48,  44,  50,  48,  55,  58
MA 1450 data 151,  50,  49,  49,  44,  48,  58, 151
MJ 1460 data  50,  49,  51,  44,  50,  53,  53,   0
```

Aligning The Commodore 1541 Disk Drive

Bob Drake
Brantford, Ont.

WARNING:
FOLLOWING THE STEPS IN THIS ARTICLE COULD RESULT IN DAMAGE TO YOUR DISK DRIVE AND WILL VOID ANY WARRANTY

The warning above is real and should be read carefully. Any time you open a sealed unit, whether a disk drive or an electric toaster, you will void any warranty on the unit. You may also damage the unit. Damage may occur regardless of the instructions you are following. You may injure yourself while attempting repairs. Electrical equipment always carries with it the risk of shock and electrocution. BUT. . .if you are careful and patient, you can align your disk drive and save up to $70.

If your disk drive is still under warranty, DON'T FIX IT. Take the drive back to your dealer for repair or replacement. Doing anything else is foolish.

The other step to take before opening the case is to make sure that indeed alignment is needed. In other words, IF IT AIN'T BROKE, DON'T FIX IT. Be sure the drive is out of alignment before you start. Use the test demo disk that comes with your drive and check the alignment using PERFORMANCE TEST. If that program says the drive is alright, then something else is wrong. If you can't get PERFORMANCE TEST to load, then try the single line programs included here. But be sure BEFORE you start. I have used these instructions to align 11 disk drives this fall in the school at which I work. Only one would not align. Later work traced its problems to a bad chip.

1. Read these instructions all the way through one or more times. Be sure you understand what they say BEFORE you start.

2. Find a dry, stable place to work. You will need a diskette that can be erased, the TEST DEMO disk that came with the drive, a medium Phillips (star head) screwdriver and a sharp pointed knife (the smaller the better).

3. Unplug the disk drive from the C64 computer and from the power source (wall plug). Turn the disk drive upside down and use the screwdriver to remove the four screws from the bottom of the unit. (You just voided the warranty!)

4. Turn the drive right way up. Don't lose any of the four screws. Gently lift the top off and set it to one side. Don't sit or step on it — that plastic shell will break easily.

5. Find the red and black wires leading from the power light onto the main circuit board you should now be looking at and unplug it. Note where it came from — you have to put it back. (Figure 1)

6. The disk drive is held into the bottom half of the case with six (6) Phillips screws driven into the plastic. Remove them and set them to one side. (Figure 1)

7. CAREFULLY, not using the circuit board, lift the disk drive out of the bottom of the case, turn the case over, and insert one of the metal tabs on the drive into a slot on the plastic bottom. The drive should stand self supported. (Figure 2)

8. Find the two screws on either side of the stepper motor. CAREFULLY use the knife to scrape away the green glue holding those screws down. Don't cut any wires accidentally or you could be buying a new drive. This is, for me, the hardest part of the job. When the glue is broken off, use your screwdriver to undo each screw slightly (no more than 2-3 full turns). Turn the stepper motor fully clockwise. Note — it only moves about a quarter of an inch at most. (Figure 3)

9. Plug the drive into the computer. Plug the power back into the drive. KEEP YOUR FINGERS OFF ANY PART THAT LOOKS LIKE IT COULD CARRY POWER. Turn on the power.

10. The drive will not be far out of alignment.

Insert your TEST DEMO disk and VALIDATE it using:

```
open 1,8,15,"v0" : close 1
```

When in alignment, the red light on the front of the drive should stay on without flickering. Gently move the stepper motor (about 1/32 inch at a time) until the red light stays on. You may have to validate the disk several times. Tighten one screw.

Format a blank disk using:

```
open 1,8,15," n0:test,tt " : close 1
```

Watch the light — it should stay on. Expect that 3 or 4 tries may be needed.

12. When you are sure the drive is aligned, run the PERFORMANCE TEST from the TEST DEMO disk to confirm the alignment. Be sure that tracks 1 and 35 will both read and write.

13. Unplug the drive.

CAREFULLY, lift the drive, turn the bottom over and place drive back into the bottom.

Replace the 6 screws holding the drive in the plastic bottom, replace the red-black wires to the green light, and put the top back on. Finally, turn the drive over and replace the 4 screws in the bottom.

14. CONGRATULATIONS! You have just aligned your 1541 disk drive.

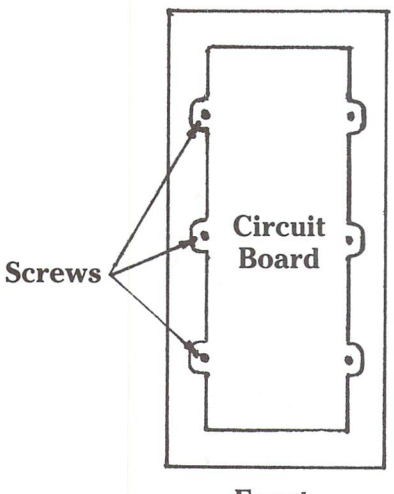


Figure 1

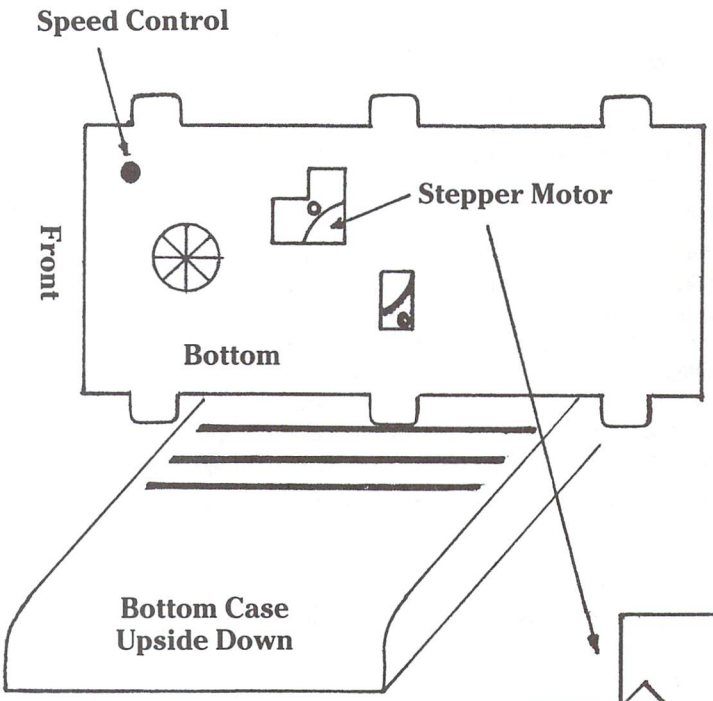


Figure 2

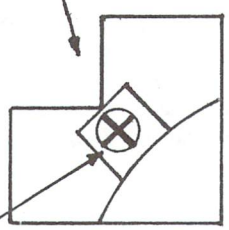
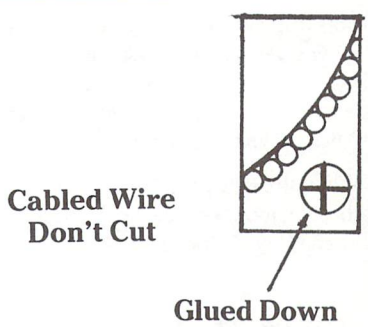


Figure 3





Super Cat

Richard Evers, Editor

Super Cat is a program written for the sole purpose of providing all that detailed directory information that we all require, but have such a hard time locating. Whatever is tucked away on the directory track of your diskette magically reappears with the use of Super Cat. For an added bonus, the output can be generated to your screen or attached printer. Look below for a quick synopsis of available data to be generated by the program :

Super Cat will display the directory track and sector, and the index into the sector that the filename resides on. It will also provide you with the usual data of block count, file name, and file type. Along with this, the track and sector of the first data block of the file itself is given. As a final bonus to REL file type users, the program will show the first side sector track and sector link, plus the record length of each record within the file. This is the sum and total of data that will become available to the Super Cat user.

Here's what to expect when generating reports with the program.

```
39:16:66    63 mlm 64.pal      prg trk 50 sec 19
```

Above is a regular display for a file, excepting REL. The first three numbers, "39:16:66" refer to the directory track, sector, and index into the sector that the filename has been located on. The next number, "63" is the block count consumed by the file on diskette. The data following is the filename followed closely by the file type. After that comes the track and sector that the first data block is held on. That's everything.

```
39:1:2    727 writer-data1    rel trk 38 sec 1 ss : trk 38 sec 8 len 254
```

This display line is from a REL type file entry. Everything is the same till you get past the track and sector link to the first data block. From "ss" on, the data printed refers to the first side sector block. In this example, the first side sector is located on track 38, sector 8, with a file length of 254 characters.

When the entire directory has been peered through by the program, the total number of blocks consumed by the programs in the directory are display, as shown below.

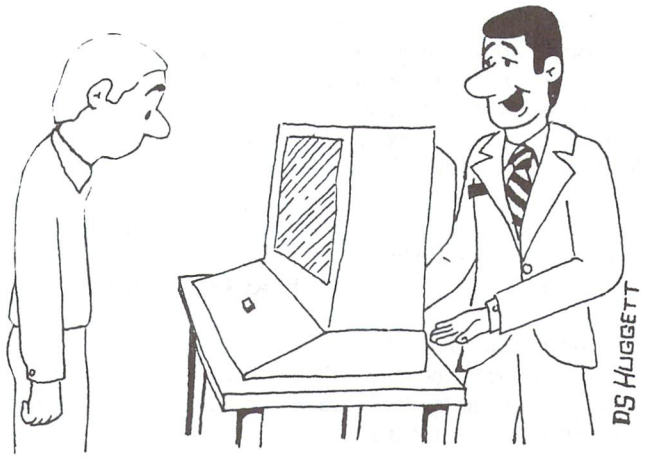
```
. . . total # blocks used : 2054 . . .
```

The program has been written to work with all Commodore floppy drives, and will even allow for a selective evaluation of the directory by using the pattern matching capabilities of the Commodore DOS.

```
IO  10 rem save "@0:super cat.bas",8
HA  100 rem ** rte/84 - a super catalog routine that supplies
        all file info
PE  110 print "** super catalog - rte/84 **"
EK  120 ft$(0)="del":ft$(1)="seq":ft$(2)="prg":
        ft$(3)="usr":ft$(4)="rel"
MK  130 print "drive type :"
MH  140 input "1) 1541/2031/4040 or 2) 8050/8250 :";ty:
        if ty<1 or ty>2 then 140
NP  150 dtrk=39: if ty=1 then dtrk=18: rem ** directory track
JM  160 input "drive #";dr: if dr>1 then 160
PH  170 print "selective directory ?"
OO  180 print "1) seq, 2) prg, 3) usr, 4) rel, 5) all"
HP  190 input sl: if sl<1 or sl>5 then 190
OO  200 sl$="all": if sl<5 then sl$=ft$(sl)
BE  210 input "3) screen or 4) printer :";dv:
        if dv<3 or dv>4 then 210
FC  220 open 15,8,15: open 5,8,5,"#": open 1,(dv)
OI  230 sec=1: ctr=0: bu=0: z$=chr$(0): sp$="  "
GL  240 print#15,"u1:"5;dr;dtrk;sec: rem ** block read
CM  250 print#15,"b-p:"5;0: ctr=0: rem ** position the
        buffer pointer
KL  260 get#5,nt$,ns$: rem ** next track/sector links
HH  270 get#5,ft$: ft=asc(ft$+z$)and15: if ft>4 then ft=0
JL  280 ft$=ft$(ft): rem ** assign file type
AK  290 get#5,t$,s$: rem ** first data trk/sector
FP  300 trk=asc(t$+z$): skt=asc(s$+z$)
DP  310 fl$="": for x=1to16: get#5,a$: fl$=fl$+
        chr$(asc(a$+z$)): next: rem filname
OB  320 get#5,sr$,ss$,rl$: rem ** first trk/sec side sector +
        length if rel file
JC  330 sr=asc(sr$+z$): ss=asc(ss$+z$): rl=asc(rl$+z$)
GC  340 get#5,gb$,gb$,gb$,gb$,gb$,gb$: rem ** 6 bytes not used
MG  350 get#5,bl$,bh$: blks=asc(bl$+z$)+256*asc(bh$+z$)
HG  360 get#5,gb$,gb$: rem ** two wasted bytes at start of
        next record
IB  370 if ft=0 then 470
NF  380 dp$=mid$(str$(dtrk),2)+":"+mid$(str$(sec),2)+":"
JE  390 dp$=dp$+mid$(str$(ctr*32+2),2)
BG  400 dp$=mid$(dp$+sp$,1,len(dp$)+(10-len(dp$)))+
        right$(sp$+str$(blks),5)+"  "
PC  410 if sl$="all" then 430: rem ** everything ok
GP  420 if sl$<>ft$ then 460 : rem ** file type incorrect
OK  430 print#1,dp$fl$"  "ft$" trk"trk"sec"skt;
CA  440 if ft$<>"rel" then print#1: goto 460
MK  450 print#1,"ss : trk"sr"sec"ss"len"rl
IB  460 bu=bu+blks: rem ** blocks used in total
HN  470 ctr=ctr+1: if ctr<8 then 270
FF  480 dtrk=asc(nt$+z$): sec=asc(ns$+z$)
FF  490 if dtrk then 240
HP  500 print#1,". . . total # blocks used :"bu". . ."
KD  510 close1: close5: close15: end
```

Software Numeric Keypad For the Commodore 64

Michael Kwun
Okemos, MI



"It's expandable to 48 keys"

The following program was written because of the wealth of Commodore 64 programs which contain DATA statements. I often overlook interesting looking programs printed in magazines due to the multitude of data statements found within and my lack of proficiency with the top row on the keyboard (the numbers). Wouldn't it be nice to have a numeric keypad?

One day while looking over a memory map searching for some ROM routines, I found the following entries:

EB81	60289	Keyboard 1 – unshifted
EBC2	60354	Keyboard 2 – shifted

Investigating these locations, it turns out that by changing their values (by transferring the ROM to RAM and switching the Kernel out), you can change the characters that are accessed by each key. The following program, "Software Numeric Keypad", was written using this knowledge.

Type it in with any machine language monitor, and save it. To use it, just load it and run. After a brief pause, the "READY." prompt will appear. The keyboard has been redefined, and at this point the program is no longer required. These are the key re-definitions:

789	(no change)	789
UIO	changes to	456
JKL	changes to	123
M,.	changes to	00,

To return the keyboard to normal, use RUN-STOP/RESTORE or POKE 1,55 (the poke is tricky to enter in numeric keypad mode). To regain access to the numeric keypad, enter POKE 1,53. While in the numeric keypad mode, the U, I, O, J, K, L and the period can be accessed by pressing SHIFT. The '<' and the '>' can be accessed by using the Commodore key. Happy typing!

Hex dump of the program:

```
.: 0801  0B 08 0A 00 9E 32 30 36
.: 0809  31 00 00 00 A9 A0 85 FC
.: 0811  A9 00 85 FB A0 00 B1 FB
.: 0819  91 FB C8 D0 F9 E6 FC A5
.: 0821  FC C9 C0 D0 F1 A9 E0 85
.: 0829  FC A0 00 B1 FB 91 FB C8
.: 0831  D0 F9 E6 FC D0 F5 A9 35
.: 0839  85 01 A2 12 BD 4C 08 9D
.: 0841  9E EB BD 5E 08 9D DF EB
.: 0849  CA D0 F1 60 34 56 39 35
.: 0851  31 30 30 32 36 4E 2B 50
.: 0859  33 2D 2C 3A 40 30 55 D6
.: 0861  29 49 4A 30 4D 4B 4F CE
.: 0869  DB D0 4C DD 2E 5B BA 2C
```

Now save from $0800 to $0871.

Disk/Extramon 64

Mike Forani
Burlington, Ontario

Disk/Extramon 64 is the all-in-one ultra-monitor for the Commodore 64. It has just about everything you could ask for, and then some. Just one problem though – it's over 7K long! And 8 pages of DATA statements not only consumes too much magazine space, but makes hand entry far too impractical. So why print instructions without the program? Mike uses standard commands and command syntax which makes the instructions practically universal for all other monitor utilities. Mike's program may have commands that the others don't, but odds are the others have none that Mike doesn't include. Two assembled versions of Disk/Extramon 64 are available on Transactor Disk #5. – M.Ed.

Embarking on an investigative journey through your 64? Yes? Then Disk/Extramon 64 is a travel companion you shouldn't be without. It's the monitor program to end all monitor programs.

A lot of you may be saying, "Why should I bother with this one?". Agreed, there are several monitor programs around, some for sale, others for free. Disk/Extramon 64 has features not found in other monitors which was going to be a strong "selling point" for the program. However, the program is now public domain so all those selling points make the fact that it's free even more attractive.

Disk/Extramon 64 has all the common machine language monitor commands such as Hunt, Assemble and Disassemble, Transfer, plus Newlocate, Interrogate, Compare, Quick Trace, and Bank Switching commands especially for the 64. Hex/Decimal conversions are in there too.

The Disk Monitor portion of the program has everything the budding young drive programmer needs for experimenting with the inner workings.

Note: The program has been tested with Commodore equipment only. I therefore cannot insure that it will work properly with non-Commodore printers, disk drives, or IEEE interfaces. (There has been some success with the 4040 dual drive and the Bus-Card II interface.)

The following is a list of the Disk/Extramon 64 commands. Some of the commands require special attention so please read on.

Affected Memory

This program is located at $1000 or at $8000 and uses 8K of memory. The page 3 vectors; IRQ, BRK, and ICRNCH are changed and the upper 5 bytes of page 0 are used by the monitor program. A number of kernal routines are also used and these will affect some zero page variables. CHRGET is used by the monitor program and this will affect the page 2 input buffer. On a break instruction before anything can change, zero page and page two are saved at $9e00 to $9fff with the $8000 version or at $2e00 to $2fff with the $1000 version of the monitor program. Therefore you will always be able to see what zero page or page two locations your program affected. Also on a break instruction the VIC chip's video and character generator registers are reset to their defaults as is the I/O port in page 0.

Note: While in the monitor all numeric input must be in hexadecimal numbers except when doing decimal to hexadecimal conversion.

Monitor Commands

DISPLAY REGISTERS: Display the current processor status.
r

DISPLAY MEMORY: Display contents of memory in hex.
m adr1 adr2 adr1 ;beginning address
adr2 ;ending address (optional)
If adr2 is left out then one line of eight bytes will be displayed.

ALTER MEMORY: Alter contents of the 64's memory
.: 1000 00 00 00 00 00 00 00 00 alter memory

ALTER REGISTERS: Alter contents of 64's processor registers.
pc irq sr ac xr yr sp
.; 1000 ea31 b0 00 00 00 ff alter processor status

GO: Begin execution of a machine language program.
g adr1 adr1 ;beginning address of execution (optional)

* If the kernal is banked out the processor status is not restored at the 'go' command and the IRQ's are disabled. Also if the address is omitted or invalid, the 64 will jump to address in the program counter.

LOAD: Load a program into the 64's memory.
l " sdr:filename " ,dn,adr1
sdr ;source drive number (optional)
dn ;device number (08 – 0f)
adr1 ;load address (optional, defaults to the disk load address)

SAVE: Save a program from 64 to disk.
s " ddr:filename " ,dn,adr1,adr2
ddr ;destination drive # (optional)
dn ;device number (08 – 0f)
adr1 ;beginning address of save
adr2 ;end address of save (last byte is saved)

EXIT: Exit the monitor to BASIC.
x * All wedges are left intact so the monitor may be reentered.

Extra Monitor Commands

MONITOR: Enter monitor from BASIC.
mon

BANK: The kernal and/or basic may be banked out of memory so that the RAM memory sitting behind it may be modified, assembled, saved, executed or traced.

bbout bank out the basic ROM to give RAM
bbin bank in the basic ROM
bkout bank out the kernal ROM to give RAM
bkin bank in the kernal ROM

TRANSFER: A portion of memory may be transferred from one memory location to another.

t adr1 adr2 adr3 adr1 ;start address
adr2 ;end address
adr3 ;beginning address of transfer

FILL: Fill a portion of memory with a given value.

f adr1 adr2 xx adr1 ;start address
adr2 ;end address
xx ;value to fill memory with

HUNT: Hunt for a string of values in a specified portion of memory.

h adr1 adr2 'string' adr1 ;start address
h adr1 adr2 xx xx xx adr2 ;end address for hunt
'string' ;characters to be searched for
xx ;hex values to be searched for
(max. length of string or bytes is 20)

COMPARE: Compare two portions of memory to each other.

c adr1 adr2 adr3 adr1 ;start address
adr2 ;end address
adr3 ;start address (second block)

* memory locations that do not compare equal will be displayed.

INTERROGATE: Display the screen printable characters along with the memory locations values.

i adr1 adr2 adr1 ;start address
adr2 ;end address (optional)

QTRACE: Trace a machine language routine and display the processor status after each instruction is executed.

q adr1 adr1 ;address to begin execution
* Pressing the 'n' key skips the trace
* Pressing the 'm' key speeds it up
* Pressing the space bar halts execution

* No separate interrupt control, unless non maskable, is allowed during the trace routine and any i/o routines may be affected. The qtrace works on an interrupting system. Interrupts occur after each instruction is executed, therefore IRQ control in the program being traced may crash the system. (CIA #1 – Timer A is used for interrupt timing.)

ASSEMBLE: Assemble a machine language program in memory. (this is a simple assembler)

a adr1 lda #$41 adr1 ;beginning address for assembly

* To end assembling a return (blank line) must be entered before doing any other operations such as altering the assembled code.

DISASSEMBLE: Disassemble hexadecimal memory location values into mnemonic op-codes with operands.

d adr1 adr2 adr1 ;start address
adr2 ;end address (optional)

* If adr2 is left out then only one op-code and its operand will be displayed.

ALTER DISASSEMBLY: Change the screen disassembly.

., 1000 20 d2 ff jsr $ffd2

* The hex values are to be changed not the mnemonics.

NEW LOCATER: Relocate a machine language program.

n adr1 adr2 offset adr3 adr4 w

adr1 ;beginning address of code to be relocated
adr2 ;end address
offset ;value to be added to absolute indexed memory locations
adr3 ;lower address limit of absolute addressed data which is to be changed
adr4 ;upper limit
w ;relocating a word table – if included

* The code to be relocated must first be transferred, this is a two step command.

DEC/HEX CONVERSION: Convert a hexadecimal number to a decimal number or a decimal number to a hexadecimal number.

***65535** ;decimal to hexadecimal
***$ffff** ;hexadecimal to decimal

KILL: The disk/extra monitor wedges are destroyed and normal basic operations may be done – the monitor may not be reentered unless you jump to the start of the program. ie. $1000 or $8000.

k * All the page three vectors used are restored.

COLD: Do a power on reset sequence.

p

Disk Monitor Commands

DIRECTORY: Do a screen list of the directory.

/ ;directory of disk
/"**m*** ;directory of files starting with the letter 'm'
/"**1:** ;directory of drive 1

READ: Read a sector from the disk to a disk buffer.

$r dd tt ss bb dd ;drive
tt ;track
ss ;sector
bb ;buffer (optional, default is 01)

WRITE: Write a disk buffer to the disk surface.

$w dd tt ss bb dd ;drive
tt ;track
ss ;sector
bb ;buffer (optional)

GET: Get disk memory to the 64's memory.

$g adr1 adr2 adr3 adr1 ;start address of get
adr2 ;end address
adr3 ;address to store at in C64

PUT: Put 64's memory to disk memory.

$p adr1 adr2 adr3 adr1 ;start address of put
adr2 ;end address
adr3 ;address to store at in drive

VIEW: View the disk drives memory.
$v adr1 adr2 adr1 ;start address
adr2 ;end address (optional)

ALTER: Alter the disk drives memory.
.$:0300 00 00 00 00 00 00 00 00

DIRECT: Send a direct command to the disk drive.
$>. . . ;any basic 2.0 disk command

* The disk status is displayed after the command is executed.

TRACE: Trace a files track and sector links and display them. (begin tracing at. . .)
$t dd tt ss bb dd ;drive
tt ;track
ss ;sector
bb ;buffer (optional)

FETCH: Fetch a sector from the disk drive surface to the 64's memory.
$f adr1 dd tt ss bb adr1 ;start address in C64
dd ;drive
tt ;track
ss ;sector
bb ;buffer (optional)

DUMP: Dump a block of the 64's memory to the disk surface.
$d adr1 dd tt ss bb adr1 ;start address in C64
dd ;drive
tt ;track
ss ;sector
bb ;buffer (optional)

CHANGE: Change the device number of the disk drive. (send to drive or just for program defaults.)
$c do dn* do ;old device number
dn ;new device number
* If the asterisk is included the change is only done in the 64's memory so that a device 09–0f may be used as a default if hard wired.

ALLOCATE: Allocate a sector as being used in the BAM.
$a dd tt ss dd ;drive
tt ;track
ss ;sector
* To de-allocate sectors use the basic 2.0 command Validate (v0)

EXECUTE: Execute disk memory.
$e adr1 adr1 ;beginning of execution

BLOCK EXECUTE: Load a sector off the disk surface into a disk buffer and execute it.
$b dd tt ss bb dd ;drive
tt ;track
ss ;sector
bb ;buffer (optional)

STATUS: Check the disk status.
$s

INTERROGATE: Display screen printable characters while displaying the memory of the disk drive.
$i adr1 adr2 **adr1 ;start address**
adr2 ;end address (optional)

Note 1 After doing any disk memory commands the drive should be initialized to avoid any unfriendly errors. ($>i0)

Note 2 An automatic scroll up and down is built into the memory display routines and the disassemble.

Note 3 Pressing the run/stop and restore keys will reset the computers page 3 vectors, this will result in the monitor not working on the scroll routines therefore the monitor must be exited and reentered at its starting point to reset the vectors once more.

Disk/Extramon 64 Quick Reference Chart

MONITOR COMMANDS

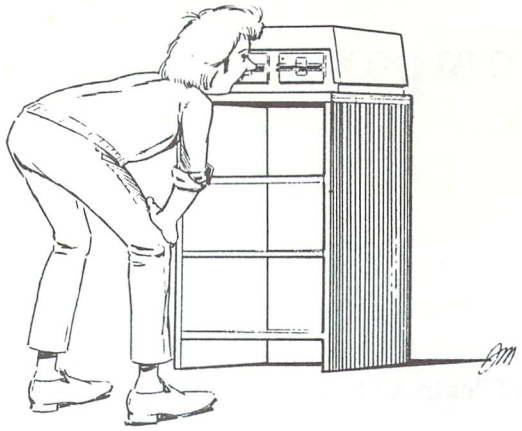
a adr1	simple assembler
bbin	bank basic in
bbout	bank basic out
bkin	bank kernal in
bkout	bank kernal out
c adr1 adr2 adr3	compare memory
d adr1 [adr2]	disassemble memory
f adr1 adr2 xx	fill memory
g [adr1]	begin execution of program
h adr1 adr2 'string'	search memory for string
h adr1 adr2 xx xx xx . . .	search memory for bytes
i adr1 [adr2]	interrogate memory
k	kill monitor wedges and exit
l " sdr:filename ",dn,[adr1]	load a file from disk
m adr1 [adr2]	display memory bytes
mon	enter monitor from basic
n adr1 adr2 offset adr3 adr4 [w]	relocate program code
p	do a power on reset sequence
q [adr1]	quick trace of program code
r	display processor registers
s " ddr:filename ",dn,adr1,adr2	save a file to disk
t adr1 adr2 adr3	transfer memory
x	exit the monitor to basic
*** xxxxx**	decimal to hex conversion
***$ xxxx**	hex to decimal conversion

DISK MONITOR COMMANDS

$a dd tt ss	allocate a sector in BAM
$b dd tt ss [bb]	block execute
$c do dn	change disk device number
$c do dn*	change disk default device #
$d adr1 dd tt ss [bb]	dump memory to disk
$e adr1	execute disk memory
$f adr1 dd tt ss [bb]	fetch sector from floppy
$g adr1 adr2 adr3	get disk memory
$i adr1 [adr2]	interrogate disk memory
$p adr1 adr2 adr3	put memory to disk memory
$r dd tt ss [bb]	read a sector to disk buffer
$s	check disk status
$t dd tt ss [bb]	trace file link pointers
$v adr1 [adr2]	view disk memory
$w dd tt ss [bb]	write a buffer to disk
$> 'string'	send disk command
/	directory

Drive Peeker: A Quick Peek Inside Your Drive

Richard T. Evers



Drive tripping has become one of North America's prime vacation pastimes over the past few years. With disk protection making its entry in such a big way, plus so many new tricks being uncovered on a daily basis, dashing about within your drive unit is the thing to do.

Well, in keeping with new trends, we bring you 'Drive Peeker', a program guaranteed to trip the light fantastic with you throughout every private part of your drive. Each and every hidden recess is no longer so, with Drive Peeker at your side. Now on to a more serious vein.

To look inside of disk memory and extract the information that lies within is not a difficult task. All that is required is to inform the drive of your intentions and proceed along with what you want to do. Before progressing on to the program, let's first explain how this can be done.

The very first thing required is to place a call to the drive along its private channel, then keep the channel open for further updates of procedures.

```
open 15,8,15
```

Once opened up, you have to inform the disk drive the reason for the call. A command as follows will do the trick

```
print#15, "m-r" chr$(ml)chr$(mh)chr$(numchr)
```

The "M-R" informs the drive that you would like to read its memory. CHR$(ML) and CHR$(MH) tells it what portion of memory you are interested in. CHR$(numchr) is an optional parameter not discussed by Commodore in past. This will inform the drive that you would like to read the value of (numchr) characters at a time. (numchr) represents a number up to 255 decimal. To get these characters, the GET#15 command is the used. If required, you can GET# more than one character at a time, ie. GET#15,a$,b$,c$,d$ . . . etc. The drive understands this and will return to you the number of characters desired.

With all disk operations using CBM equipment, a zero byte cannot be retrieved at all. You can write it to disk and it will actually make it there in one piece. But when you try to read it back, the drive unit eats it and gives you nothing in return. Something like a users fee. Well, whenever reading data from disk, a test should be made to see if the string received is actually there. If not, assign it with a value of chr$(0), and your programs will be happier.

Once you have that byte, you can do whatever you please. In the program below, I have used it for one specific purpose, to show you what is hiding in your unit. Once the data is received, its ASCII value is OR'd with 64 to make it nice to work with, then it is added to a string. From there, the original data byte received is converted to hexadecimal and is printed out. A maximum of 8 bytes will be displayed per line, then the string of the CHR$(value or 64) will be printed. This will help you at times when you know you're peering at a .byte table.

Every time a line is dropped, the current location is memory is printed at the beginning of the new line in hexadecimal. In this way, you always know where the data is coming from, just in case you want to find it back later on. As an extra bonus, the program has been written to allow output to be directed to the screen or printer, as you please. With everything taken into consideration, the routine isn't bad for the amount of work it's going to cost you keying it in. And, as usual, you will have learned but another new programming trick, to add to your already overflowing collection.

```
100 rem : save "@0:drive peeker",8:verify "0:drive peeker",8
105 rem : ** drive peeker - rte/84 **
110 rem : ** will scout about within your drive & report back to you **
115 print "** drive peeker **"
120 hx$ = "0123456789abcdef"
125 input "hex : start, end ";s$,e$
130 va$ = s$ : gosub215 : s = val(va$) :rem convert start to dec
135 va$ = e$ : gosub215 : e = val(va$) :rem convert end  to dec
140 input "s) screen : p) printer ";sp$
145 dv = 3 : if sp$ = "p" then dv = 4
150 open 1,(dv)
155 open 15,8,15
160 for pk = s to e step 8 : v$ = ""
165 mh% = pk/256 : ml = pk-mh%*256
170 print#15, "m-r" chr$(ml)chr$(mh%)chr$(8) :rem read in 8 chars
175 flag = 1 : v = mh% : gosub 190 : print#1,ht$; : v = ml : gosub 190
      : print#1,ht$ "  "; : flag = 0
180 for x = 0 to 7
185 get#15,a$ : v = asc(a$ + chr$(0)) : v$ = v$ + chr$(vor64)
190 h% = v/16 : lo = v-h%*16
195 ht$ = mid$(hx$,h% + 1,1) + mid$(hx$,lo + 1,1) : if flag then return
200 print#1,ht$ " ";
205 next x : print#1,v$ : next pk : close1 : close 15 : end
210 rem hex - dec
215 ln = len(va$) : f = 0
220 forx = 0toln-1 : v = asc(mid$(va$,ln-x,1))-48 : y = abs(v>9)
225 f = f + ((v-(y*7))*16↑x) : next : va$ = mid$(str$(f),2) : return
```

File Compare

Richard Evers

Compare Disk Files In A Flash

A while ago I took it upon myself to convert the PET resident monitor for the Commodore 64. Considering how many MLM's are already out there, this was a pretty useless task. Needless to say, you will never actually see the completed version in the pages of our magazine. To get back to the story, troubles occured immediately with my assembled pseudo source. To get the source, I simply SAVEd the MLM from the PET in memory to disk, then disassembled it with a disk unassembler program. After a few mods to this apparent source, it was assembled, just to check if everything was OK. With a crash and heave my hopes dissolved. It wasn't quite right. LOAD and SAVE were OK, so were all but one of the other commands. G, the GO command, would crash the machine by trampling back into BASIC with the decimal flag set. It came close to driving me crazy because the machine would have to be powered down after every attempt at correcting the error.

If a comparison of code was made between the original MLM in the PET, and my pseudo assembled code, the trouble could quickly be found. The incorrect or missing bytes would easily be pointed out, with the right program. An so, File Compare #1, a BASIC version, was conceived.

In reality the BASIC program took a very short period of time to write. How much effort does it take to OPEN two files for a read, read in each byte in tandom, compare them, then flag to the screen when something wasn't quite right. Not that complex. Needless to say, my 5 minutes of tedious labour found the problem. A byte was mixed up in my disassembly, therefore the new code would always err out. Once the byte was fixed, the code was great. A happy ending. And so, the BASIC version of File Convert became part of my already overflowing archives of limited use programs.

Enter the utilities issue. A perfect occasion for the rebirth of a concept long forgotten. With a burst of BAID, and a final lunge with PAL, File Compare, machine code version, was born. No longer the boring little BASIC ditti that took forever to finish. This one goes like a bad odor on a windy day, which means, it goes like stink. For an encore it will generate its report to your screen or printer. The best part of this one is the fact that, as Chris Zamara says, it doesn't do you any favours. It simply does its job without messing up your computer in the process. A simple bit of code for a simple task.

The programs listed below are for the sole purpose of creating a program on diskette with a single BASIC line to SYS to the start address. Once created, LOAD the new program in and RUN it as a normal BASIC program. The code has been written to start at $0401 for the PET/CBM and $0801 for the C64. The code cannot be relocated without reassembling the source.

Once up and running, the first prompt will ask you for the device # in which to output the report to, either 3 (screen) or 4 (printer). Answer the question, hit return and wait for the next question. The next prompt,

" dr#:filename > "

will appear. Type in the name of one of the files in which to compare then hit return. The prompt will then reappear immediately. Reply this time with the name of the second file. The compare begins.

The display is formatted to first show the index (ie. 0–max) into the file that the mismatch was found, followed by the byte value found in the first file, then the byte value from the second file. Everything is displayed in hex. If no errors are found, the files will simply be read through without any great excitement. When complete, it will return to perform a BASIC warm start back to READY mode. Very simple.

If you specify a file that doesn't exist, the program will understand and close everything up, then return to BASIC without harm. If you find that you have to STOP the program, for whatever reason at all, press the STOP key, and control of your computer will be passed back to you, with all files correctly closed up.

Quick note before the code. ST is checked after the byte is taken from the second file specified. If you know that one file will be larger than the other, and don't want to extend beyond the limit of the smaller one, then specify the larger file first, the smaller one last. This will save a display full of $0D's from the small file in comparsion to the larger on. The two files will not match too well at this point, so you will generate a report of garbage.

Please remember to SAVE the program(s) below before running them. Even though it creates another program for you, it won't hurt to have the generator around in case of an error.

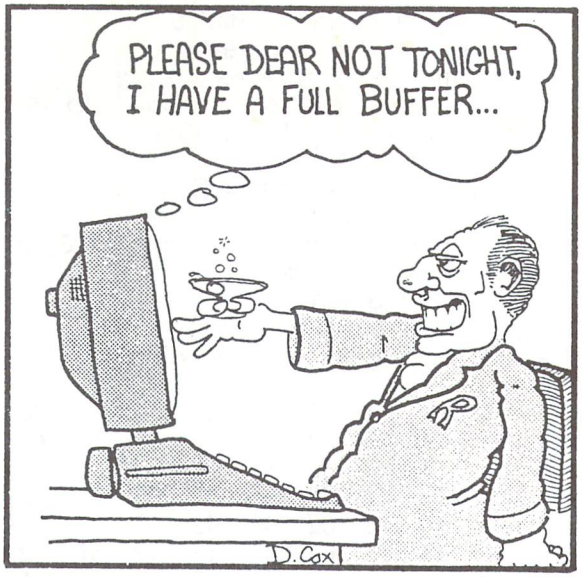
File Compare: BASIC 4.0 Version

```
CP 10 rem  save " '0:filcomp pet.dat " ,8
MC 100 rem ** rte/84 - data to create
       'file compare 4.0' on diskette as prg
FI 110 input " drive #, new program name " ;dr$,fl$
HF 120 open 15,8,15: open 5,8,5,(dr$) + " : "
       + fl$ + " ,p,w "
PI 130 input#15,e,e$,b,c: if e then close 15:
       print e,e$,b,c: stop
DE 140 for j = 1025 to 1409 : read x: print#5,chr$(x);:
       ch = ch + x: next: close5
DH 150 if ch<>39775 then
       print " checksum error " : end
IP 160 print " ** program complete ** " : end
OB 170 :
MC 180 data   1,   4,  37,   4,  10,   0, 158,  32
KB 190 data  49,  48,  54,  51,  32,  58,  32,  42
GC 200 data  42,  32,  70,  73,  76,  69,  32,  67
FF 210 data  79,  77,  80,  65,  82,  69,  32,  52
HH 220 data  46,  48,  32,  42,  42,   0,   0,   0
NE 230 data 160,   5, 169,  71,  32,  29, 187,  32
BI 240 data 207, 255,  56, 233,  48, 133,  90, 201
NB 250 data   3,  48, 244, 233,   5, 176, 240, 169
JP 260 data  13,  32, 210, 255, 169,   0, 133, 209
OE 270 data 169,   1, 133, 210, 165,  90, 133, 212
JG 280 data 169, 255, 133, 211,  32,  99, 245, 169
ED 290 data   0, 133,  87, 133,  88, 162,   5, 160
FC 300 data   5, 169,  97,  32,  11,   5, 162,   6
MM 310 data 160,   5, 169,  97,  32,  11,   5, 165
OB 320 data 155, 201, 239, 208,   3,  76, 246,   4
FG 330 data 162,   5,  32, 198, 255,  32, 207, 255
DE 340 data 133,  89,  32, 204, 255, 162,   6,  32
JL 350 data 198, 255,  32, 207, 255,  72, 165, 150
HO 360 data 133,  91,  32, 204, 255, 104, 197,  89
PH 370 data 240,  78,  72, 162,   1,  32, 201, 255
MJ 380 data 169,  36,  32, 210, 255, 165,  88,  32
HO 390 data  49,   5, 165,  88,  32,  61,   5, 165
JA 400 data  87,  32,  49,   5, 165,  87,  32,  61
OJ 410 data   5, 169,  32,  32, 210, 255, 169,  36
HA 420 data  32, 210, 255, 165,  89,  32,  49,   5
DI 430 data 165,  89,  32,  61,   5, 169,  32,  32
IM 440 data 210, 255, 169,  36,  32, 210, 255, 104
BM 450 data  72,  32,  49,   5, 104,  32,  61,   5
KC 460 data 169,  13,  32, 210, 255,  32, 204, 255
FC 470 data 230,  87, 165,  87, 208,   2, 230,  88
AE 480 data 165,  91, 208,   3,  76, 110,   4, 169
AP 490 data   1,  32, 226, 242, 169,   5,  32, 226
BA 500 data 242, 169,   6,  32, 226, 242,  32, 204
KE 510 data 255,  76, 255, 179, 134, 210, 134, 211
HN 520 data  32,  29, 187,  32, 226, 180, 162,   0
HC 530 data 189,   0,   2, 240,   3, 232, 208, 248
GC 540 data 134, 209, 169,   0, 133, 218, 169,   2
HE 550 data 133, 219, 169,   8, 133, 212,  32,  99
PJ 560 data 245,  96,  74,  74,  74,  74, 170, 189
CF 570 data 112,   5,  32, 210, 255,  96,  41,  15
CH 580 data 170, 189, 112,   5,  32, 210, 255,  96
BL 590 data  51,  41,  32,  83,  67,  82,  69,  69
HK 600 data  78,  32,  79,  82,  32,  52,  41,  32
IM 610 data  80,  82,  73,  78,  84,  69,  82,  32
IL 620 data  58,   0,  68,  82,  35,  58,  70,  73
FB 630 data  76,  69,  78,  65,  77,  69,  32,  62
CH 640 data   0,  48,  49,  50,  51,  52,  53,  54
JB 650 data  55,  56,  57,  65,  66,  67,  68,  69
PP 660 data  70
```

File Compare: Commodore 64 Version

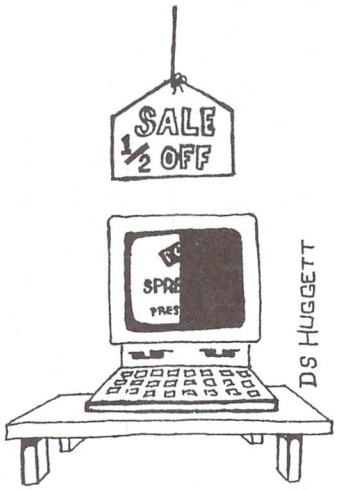
```
LC 10 rem  save " @0:filcomp c64.dat " ,8
IH 100 rem ** rte/84 - data to create
       'file compare c64' on diskette as prg
FI 110 input " drive #, new program name " ;dr$,fl$
HF 120 open 15,8,15: open 5,8,5,(dr$) + " : "
       + fl$ + " ,p,w "
PI 130 input#15,e,e$,b,c: if e then close 15:
       print e,e$,b,c: stop
CG 140 for j = 2049 to 2433 : read x: print#5,chr$(x);:
       ch = ch + x: next: close5
CG 150 if ch<>38861 then
       print " checksum error " : end
IP 160 print " ** program complete ** " : end
OB 170 :
ID 180 data   1,   8,  37,   8,  10,   0, 158,  32
BA 190 data  50,  48,  56,  55,  32,  58,  32,  42
GC 200 data  42,  32,  70,  73,  76,  69,  32,  67
OF 210 data  79,  77,  80,  65,  82,  69,  32,  67
CG 220 data  54,  52,  32,  42,  42,   0,   0,   0
NB 230 data 160,   9, 169,  71,  32,  30, 171,  32
BI 240 data 207, 255,  56, 233,  48, 133,  90, 201
NB 250 data   3,  48, 244, 233,   5, 176, 240, 169
AB 260 data  13,  32, 210, 255, 169,   0, 133, 183
CH 270 data 169,   1, 133, 184, 165,  90, 133, 186
NG 280 data 169, 255, 133, 185,  32,  74, 243, 169
ED 290 data   0, 133,  87, 133,  88, 162,   5, 160
JD 300 data   9, 169,  97,  32,  11,   9, 162,   6
AO 310 data 160,   9, 169,  97,  32,  11,   9, 165
BC 320 data 145, 201, 127, 208,   3,  76, 246,   8
FG 330 data 162,   5,  32, 198, 255,  32, 207, 255
DE 340 data 133,  89,  32, 204, 255, 162,   6,  32
DM 350 data 198, 255,  32, 207, 255,  72, 165, 144
HO 360 data 133,  91,  32, 204, 255, 104, 197,  89
PH 370 data 240,  78,  72, 162,   1,  32, 201, 255
MJ 380 data 169,  36,  32, 210, 255, 165,  88,  32
DP 390 data  49,   9, 165,  88,  32,  61,   9, 165
JB 400 data  87,  32,  49,   9, 165,  87,  32,  61
KK 410 data   9, 169,  32,  32, 210, 255, 169,  36
DB 420 data  32, 210, 255, 165,  89,  32,  49,   9
DJ 430 data 165,  89,  32,  61,   9, 169,  32,  32
IM 440 data 210, 255, 169,  36,  32, 210, 255, 104
BO 450 data  72,  32,  49,   9, 104,  32,  61,   9
KC 460 data 169,  13,  32, 210, 255,  32, 204, 255
FC 470 data 230,  87, 165,  87, 208,   2, 230,  88
ME 480 data 165,  91, 208,   3,  76, 110,   8, 169
MO 490 data   1,  32, 145, 242, 169,   5,  32, 145
NP 500 data 242, 169,   6,  32, 145, 242,  32, 204
PG 510 data 255,  76, 116, 164, 134, 184, 134, 185
NC 520 data  32,  30, 171,  32,  96, 165, 162,   0
HC 530 data 189,   0,   2, 240,   3, 232, 208, 248
HE 540 data 134, 183, 169,   0, 133, 187, 169,   2
LG 550 data 133, 188, 169,   8, 133, 186,  32,  74
NJ 560 data 243,  96,  74,  74,  74,  74, 170, 189
OF 570 data 112,   9,  32, 210, 255,  96,  41,  15
OH 580 data 170, 189, 112,   9,  32, 210, 255,  96
BL 590 data  51,  41,  32,  83,  67,  82,  69,  69
HK 600 data  78,  32,  79,  82,  32,  52,  41,  32
IM 610 data  80,  82,  73,  78,  84,  69,  82,  32
IL 620 data  58,   0,  68,  82,  35,  58,  70,  73
FB 630 data  76,  69,  78,  65,  77,  69,  32,  62
CH 640 data   0,  48,  49,  50,  51,  52,  53,  54
JB 650 data  55,  56,  57,  65,  66,  67,  68,  69
PP 660 data  70
```





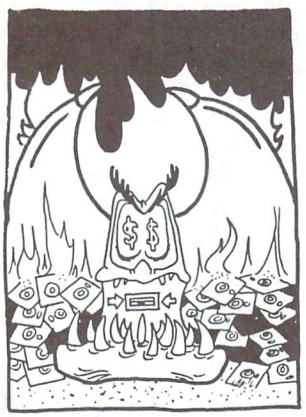


THINGS THAT GO "CLICK CLICK" IN THE NIGHT

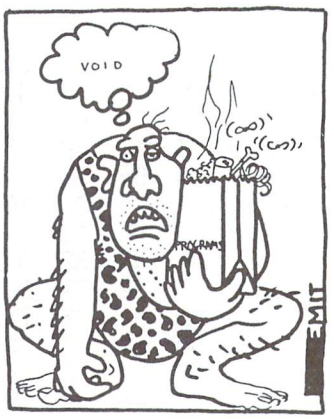




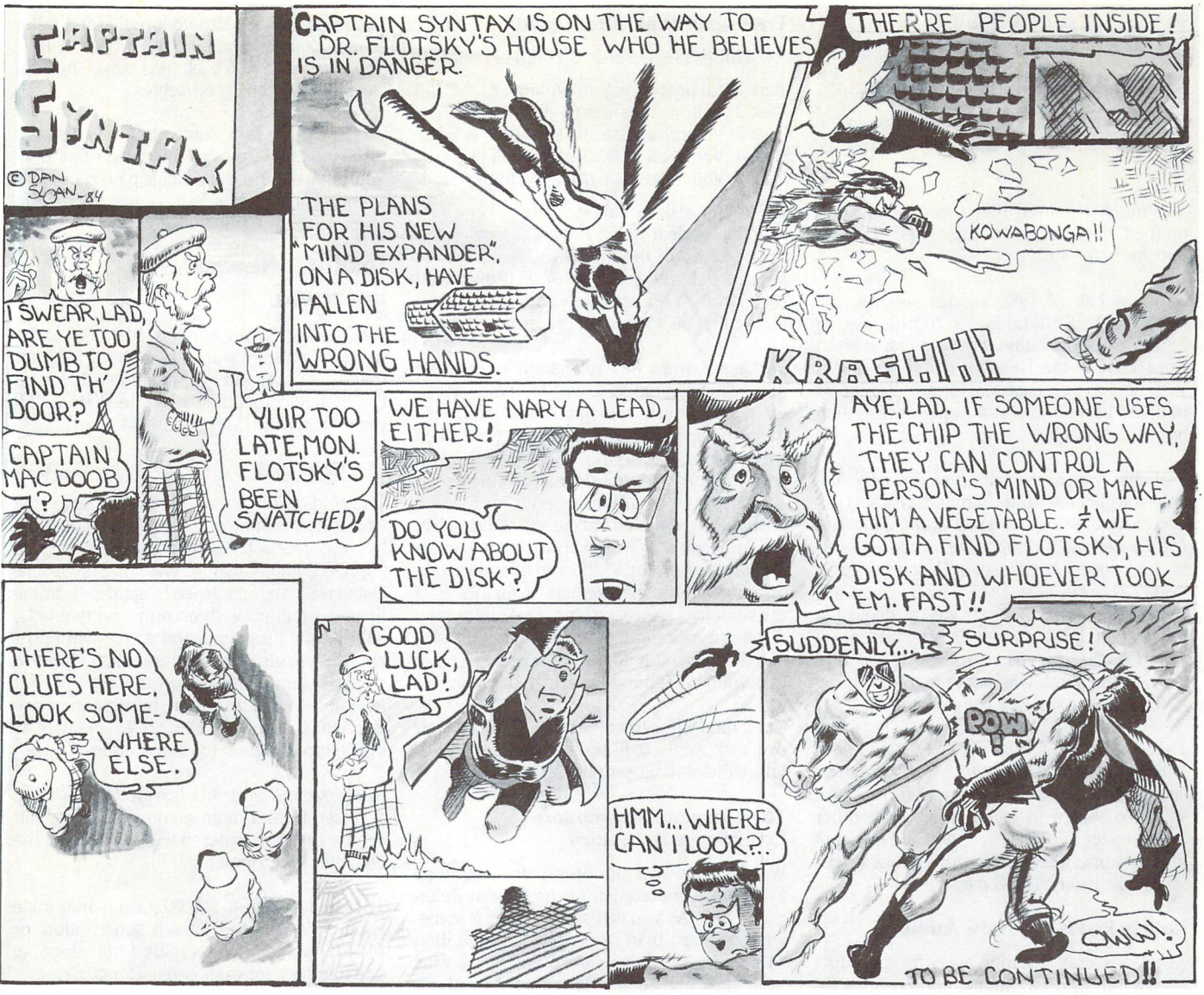
LAST CHANCE! PRINT!

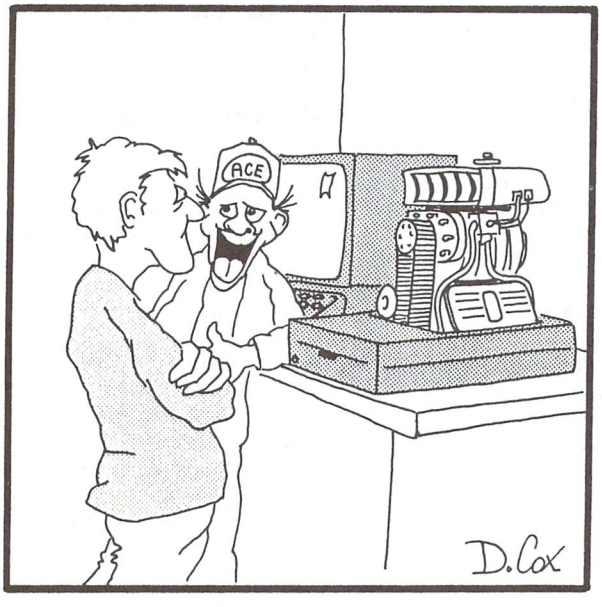


HOW PROGRAMMERS SEE COMPUTERS

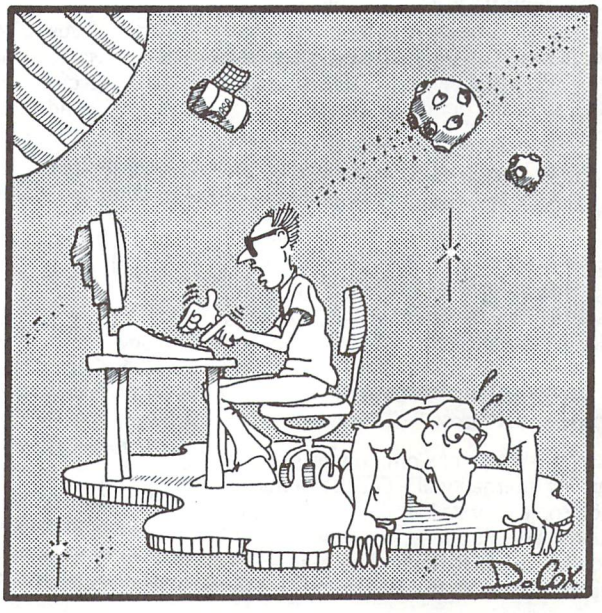


HOW COMPUTERS SEE PROGRAMMERS





It'll Backup a Disk in 6 Nanoseconds!



My Time Warp program is almost done.
I just have to de–bug the re–entry routine.

News BRK

New 16–bit Commodores for 1985

Commodore plans to regain its position in the business market with three new entries for 1985.

The much–rumored IBM compatible based on the Canadian Hyperion is slated for introduction in the spring of 1985.

The first half of 1985 should also see the release of Commodore's multi–tasking Z8000–based machine that was first seen in April 1984 at the Hanover computer fair in West Germany. This Machine will support multiple users and run an operating system based on UNIX Version 7.

Probably the most exciting of the new machines will come from Commodore's acquisition of Amega Corp. of Santa Clara, Calif. The Amiga–derived Commodore is expected to be introduced at Chicago's Consumer Electronics Show in June 1985. The machine will have a 68000 processor, as well as dedicated processors for animation, graphics, and sound (including voice synthesis). Graphics are a strong point, with super high resolution AND 32 colours from a palette of 3,000.

There's also the C128, a souped up 64 with 128K RAM and 80 column monochrome or colour display output. A portable machine was also shown to compete with the other LCD "lap-tops". For a more detailed report of the Consumer Electronics show in Las Vegas, see the February TPUG magazine.

Note to Product Review Authors

The Transactor is getting away from product reviews and prints product *comparisons* instead. If you have a product review you'd like to submit, contact TPUG magazine. Nick Sullivan, a regular contributor to The Transactor (Author of the TransBASIC series) is now the editor of TPUG and tells us he'd be happy to consider software or hardware reviews for publication.

If you'd like to write a product comparison for the Transactor, let us know the subject in advance before submitting the article. An example of a good product comparison would be one comparing the popular sketching programs like Flexidraw, Koala Pad, Doodler, Textsketch, CADpic, etc.

Over 41,000 Attend World of Commodore II Show in Toronto

The second World of Commodore show at Toronto's International Centre attracted 41,516 people over its four day run. Commodore had a large display at the show, but held back on the introduction of their new 16–bit machines, displaying mainly the +4 and 16.

The winner of the draw for the PET 2001 (a collector's item) at The Transactor booth was Bill Taylor of Acton, Ontario.

Transactor Disk Offer Update

As of this issue there are 5 Transactor Disks:

Disk 1: All programs from Volume 4
Disk 2: Volume 5, Issues 01–03
Disk 3: Volume 5, Issue 04 (Business & Ed.)
Disk 4: Vol. 5, Issue 05 (Hardware/Periphs.)
Disk 5: Vol. 5, Issue 06 (Aids & Utilities)

Transactor disks are now available on a subscription basis through the order form in the centrespread of the magazine. Disks can be purchased individually for $7.95 (Cdn.) each; the extra two dollars that was to be charged for the first disk has been dropped.

Perhaps a word of explanation is in order here. The original idea was to charge $9.95 for the first disk you purchased, and we'd make up a mailer for that disk (and each subsequent one) which contained your name, address, paid postage, and a two dollar off coupon for future disks. As it turned out, post office regulations nixed the mailer idea, so we decided to just send disks out on an individual basis for $7.95, and offer disk subscriptions. The subscription is mainly as a convenience, since our pricing philosophy dictates a rock–bottom price for single disks, and the discount for a subscription rather than individual purchase isn't that great.

To anyone who already sent in $9.95 for their first disk we'll credit the two dollars toward future disks or subscriptions.

The Complete Commodore Inner Space Anthology

We are now taking orders for the long awaited second edition of the Special Reference Issue. As you can see, the title is somewhat different than its predecessor, but then so is the inside. Of course most of the material from the first edition is included, with as much again added. See the back cover ad.

The price? Just $12.95! Originally the price was projected at around $25 dollars. Two reasons account for the difference. First, the disk we intended as part of the package will now be made available separately (details next issue). Secondly, we have decided to publish the book on our own. Previously we had considered releasing the book to an outside publisher but by doing it ourselves the price can brought down substantially.

The Complete Commodore Inner Space Anthology is currently available by mail order only through The Transactor. The easiest way to order is with the postage paid reply card at the center of the magazine. Mark the card appropriately and don't forget your postal/zip code. If you're paying by charge card, please include the expiry date. If you're sending a cheque, you can tape the postage paid card to the outside of an envelope. Please allow 4 weeks for delivery.

Autographed by Jim Butterfield

At this moment we have 50 autographed copies of Jim Butterfield's book, "Machine Language for the Commodore 64, and other Commodore Computers". 49 of them will be available for $17.95 each (no taxes). The 50th we'd like to keep for ourselves.

To get one, act fast. Remember, this notice is printed 63,000 times so they won't last long. You can even use the postage paid reply card at the center of this issue – just be sure to specify the book title on the card.

Product News:

IBM COMAL

IBM hasn't officially announced it yet, but the word is that they will soon offer COMAL for the PC, XT, AT, and PCjr machines. It will be completely compatible with the COMAL 2.0 cartridge for the Commodore 64.

The Gold Disk

The gold disk is a monthly magazine for the C64 in disk format which contains high quality software. Each issue has a feature program (eg. December's feature was an easy to use Database), an arcade–style game, a home finance program with accompanying article, educational programs, and a crossword puzzle. There is also a regular graphics column, a music column, and programming tips. An assembler, word processor, information management system, and FORTH are planned as the feature program for future issues.

All programs and articles on the disk are accessed from a main menu, and the menu can be easily re–entered without leaving the gold disk environment.

The Gold disk costs $15.00 (Cdn.) for a single issue, $70.00 for a 6 month subscription, or $127.00 for 12 months (plus $1.00 shipping and handling for each issue). Contact:

The Gold Disk
2179 Dunwin Drive, #6
Mississauga, Ont.
Canada L5L 1X3

Quick Data Drive For C64 and VIC-20

The Quick Data drive from Entrepo Inc. of Sunnyvale, CA is a high–speed replacement for the C2N datasette unit. It works with tiny tape cartridges called "microwafers" which can store from 16K to 128K of data. An operating system which comes with the drive called QOS (Quick Operating System) allows normal BASIC I/O commands (OPEN, SAVE, LOAD, etc.). Files are stored sequentially on the wafer, but the drive will search a given filename, appearing like a 1541 disk drive to the user. Average access time to locate a file on a 65K microwafer is 25 seconds. Once located, data communication rates are claimed to be 15 times faster than the C2N datasette (that's faster than a 1541 disk drive!).

Price for the drive and software on microwafer is $129.95. Microwafers cost approximately $5.00 depending on storage capacity.

Entrepo
1294 Lawrence Station Road
Sunnyvale, CA
USA 94089

Software Developers Newsletter

The Software Developers Association is a non-profit association of computer software developers, and others in related areas, who have joined together to strengthen software development in Canada.

The Software Developers Association Newsletter is a monthly publication produced and distributed by the Software Developers Association for our members and friends of a developing Canadian software industry. For further information, please contact, Bob Bruce, Chairman, Software Developers Association.

Computer Software And Human Development Conference

The impact of computer software on the future of business and education will be examined at an international conference sponsored by the Ontario Software Developers Association in conjunction with the Third Annual Software Panorama at the Royal York Hotel in Toronto, May 22-24, 1985.

Senior business, government and academic representatives from 160 countries are being invited to attend the Computer Software and Human Development Conference which will examine the many dimensions of software development and its impact on business, education, health and agriculture. The Software Panorama will also provide an opportunity for software developers to demonstrate and market their products. The Canadian software industry is estimated to consist of a thousand firms with estimated sales at $1 billion. The market is expected to grow to $5 billion by the end of the decade.

The Ontario Software Developers have established an advisory committee of senior industry representatives to make recommendations on various aspects of the conference coordination. Focusing on the School of the Future, Office of the Future, Hospital of the Future and the Farm of the Future, the exhibitors and conference are expected to underline current and future oriented developments in software. While the conference is expected to centre on trends and developments in the industry, it will also examine emerging opportunities and the adjustments required in the quickly changing industry. Please contact, Reuben Lando, Conference Coordinator.

The Software Developers Association
185 Bloor St. East
Suite 500
Toronto, ON
416 922-1153

Commodore Now Provides American Educational Software in Canada.

Commodore is now offering software across Canada from American Educational Computer, Inc., one of the leading educational software firms in North America.

"AEC's educational publishing experience has led to the development of the most extensive collection of classroom-related programs available, including phonics, word skills, reading comprehension, vocabulary skills and world geography," said Richard McIntyre, Vice-President - Sales, Commodore, Canada.

"Unlike many producers of 'educational' software who have entered the market following the softening of the game market, AEC was founded by experienced educational publishers who saw a genuine need for educational software that directly related to the classroom experience. All AEC software products follow standard school curriculum material rather than a game format and are designed to help student improve classroom performance," he said.

Commodore is looking to increased growth from this type of product, according to McIntyre. Recent research by Future Computing Inc. shows that while personal computer software sales will grow at an annual compounded rate of 68 percent through 1987, educational software will grow at a 71 percent rate during the same period and that the home educational software portion will be about 70 percent.

All AEC software is compatible with the Commodore 64. Initial products include the EASYREADER series and the MATCHMAKER series. The programs are teacher-designed and are grade-level oriented to help the child all the way through school. By paralleling the classroom experience, the software consistently teaches lessons tailored to the child's needs. All programs progress at the user's own pace, are easy to use and require no previous computer experience.

EASYREADER presents phonics and word analysis skills with high resolution graphics and full-colour animation and most importantly, correlates with standard school reading programs.

MATCHMAKER retains the format system of the school version which, in the home, allows parents to become more active in the child's learning process, through the interactive format.

Donald R. Thompson, AEC Vice-President & Director of Consumer Products Division, said, "More important than fun being written into the program is the fact that satisfaction and reward come from success. Our line is programmed so the child achieves a high level of success. Too many other educational programs are really just games that do not relate enough to classroom work."

"AEC programs do contain games, but only as rewards for learning achievement," said Thompson. For example, once the student completes the objective in MATCHMAKER'S geography program, he or she can play an exciting game, which helps to encourage and motivate. The focus is strictly on learning.

"AEC software has an important advantage - its approach has been student-tested under actual classroom conditions, so we know it keeps the child's interest while it teaches," he said. For more information, please contact:

Richard McIntyre
Commodore Business Machines, Ltd.
3370 Pharmacy Ave.
Agincourt, On
M1W 2K4 416 499-4292

LAMP

LAMP (Literature Analysis of Microcomputer Publications) has made available for sale the Annual Cumulative Edition for 1983, marking the first complete year of publication for this international index. A bi-monthly journal, LAMP presently indexes 130 periodicals which deal exclusively with the field of microcomputers. This important publication is the most comprehensive, single source for information on microcomputers as they relate to business, education, the arts, social and physical sciences.

The annual cumulative edition is printed in two volumes and encompasses thousands of subject and author entries, thousands of reviews of books and periodicals, hardware and computer systems, educational courseware and video games, and information on all phases of microcomputers.

The 1983 year-end issue is available to non-subscribers at $69.95. Or subscribe to LAMP for 1984 at the regular annual rate of $89.95 and take advantage of the special offer of $39.95 for the 1983 cumulative edition. For a brochure describing LAMP and further information on the hard-cover or microfiche editions call toll-free 800 526-9042 or write:

LAMP/Soft Images
Brochure Department
200 Route 17
Mahwah, N.J. 07430

Porthole

Porthole announces the modern computer magazine published entirely on disk for the Commodore 64 and VIC-20. The first issue is scheduled for January 1985 (or maybe December 84) and will be issued six items a year. Porthole will have all the features you have come to expect from a complete com-

puter magazine. Feature Articles, Games, Reviews, Education, Programming Tutorials, Letters and, yes, even advertising and new product announcements. But one thing you will never have to do again is to key in a program. Each issue will contain ready to run programs selected for a wide range of user interests. These will be programs that you will want to back up on your own disks – and Porthole will let you do it.

Porthole is not yet available at your local news stand. All sales are by mail. The single issue price is $10.00 postpaid. Send your orders to:

Porthole
P.O. Box 135
Kerby, OR 97531

WANTED – Programmers, Authors, Contributors, and most of all, Readers!

SPECIAL OFFER – Send a formatted 1541 disk with return postage to Porthole. By swift return mail Porthole will return the disk loaded with information on how to contribute to Porthole. But you already know! It's this easy. Write a BASIC "program" like this:

```
0 " my name is jane. porthole sounds
1 " terrific. this isn't the best word
2 " processor but it works! do you
3 " have something better? return by
4 " disk to 555 pal road, lincoln, ne
```

Don't goof up. Send return postage or Porthole gets a free disk. Sorry, letters without a disk must wait on the poop deck for the galley to empty.

EXTRA SPECIAL OFFER – Send a disk and $5.00 and ship out in steerage. Enjoy the view everyone else will see thru the Porthole window at half the price. This offer may not be repeated.

Don't miss this adventure in computing. Let's do it right – throw away those pencils and paper. Start your drives and enjoy the new spirit of magnetic publishing! For more details, contact:

Raymond Quiring
Porthole Disk
P.O. Box 135
Kerby, OR 97531
502 592–4594

New Income Tax Program For Commodore PLUS/4

Taxaid Software, Inc. has released a new edition of the "TAXAID" income tax preparation program for the Commodore PLUS/4 computer. TAXAID programs have been available for other Commodore computers since 1981. The PLUS/4 edition was written by the experienced tax accountants and is designed for home use.

TAXAID is easy to use with a detailed manual that leads the user step by step through the data entry. The program is menu driven with advanced editing features that allow the user to make changes and revisions at any time during the data entry process. Data files can be saved and reloaded at any stage of the program. Calculations are automatic and all tax tables, including income averaging are built in. TAXAID will prepare any IRS form 1040. The program features computer generated forms for Schedule A, B, C, G, W, and Form 2441 as well as a complete listing of pages 1 & 2 of the 1040 Form.

The results can be directed to the monitor or the printer. Low cost updates for future tax years are published every year.

TAXAID is available on disk or tape for the Commodore PLUS/4 at a cost of $29.95. For more information, contact:

Taxaid Software, Inc.
606 Second Ave. S.W.
Two Harbors, MN 55616
217 734–5012 218 834–3600

INFOQUICK Bulletin Board for the Commodore 64

Expandability: Can run on a single disk drive or as many as 4 dual drives. Up to 16 sub-boards for messages (including 4 privileged areas). Files library can be expanded to an unlimited number of sub-directories ("downloading"). Up to 400 users, 800 messages.

Speed: 100% machine language. Entire program and all menus are loaded once – no time consuming "chaining". Full type-ahead. Menus cut off instantly when commands are typed. Abbreviated menu option.

Flexibility: Runs on all Commodore and MSD disk drives. Runs on all the most popular modems. Can be run so users create their own fully-validated accounts or with user-created accounts requiring SYSOP validation or with all new unvalidated users on a single "generic" account. Can use standard ASCII or Commodore ASCII. Messages are automatically reformatted for the terminal of the user reading them. Changeable SYSOP name. SYSOP-selectable time limits on inactivity and on the total length of calls. SYSOP-definable welcome and warning messages. Uploads or downloads can be temporarily disabled. Optional remote SYSOP usage – can be easily enabled and disabled. 300/1200 baud operation. 70-plus page detailed manual. File transfers either as straight text or using the industry-standard XMODEM protocol with error checking. Read/scan messages since last call, forwards, reverse, by sender, by recipient; optional mark during scan for later read. Find-and-replace text editor for message entering SYSOP designated "bulletins" shown to each user or to a subset of the users. Convenient reply-delete option after reading each message. SYSOP can change public messages to private, forward a copy of a message (optionally editing it first), re-assign a message to a different sub-board. Real-time log of logins, logoffs, and file transfers. Multi-line descriptions of each file and sub-directory. "Privileged" messages and files completely invisible. Optional transcript to printer. Scan userlist by users' interests, location. SYSOP/user chat initiated by either SYSOP or by user.

INFOQUICK – Your once and future BBS. Suggested retail $139 (U.S.). Ask about SYSOP referral rebate. Dealer inquiries welcome.

For more INFO QUICK-ly call 617 547-0340 or contact these operational INFOQUICK bulletin boards:

617 823-6140 MASSPET II
203 397-3381 MicroTechnic Solutions
203 481-9974 SAIL Software

The SMART 64 Terminal +4

The SMART 64 Terminal +4 is a greatly enhanced version of this versatile terminal communications package for the Commodore 64 which is already in home education and business environments. New features include VT52/VT100 emulation when appropriately configured with 80 column hardware, XMODEM file protocol for direct-disk transfers of programs and text, HELP screens for instant reference, 300/1200 baud full-speed downloading, direct printing of the 28K memory buffer, and an expanded status line. Convenience items such as software alphabetic shift, linefeed toggle, word-wrap control, time-of-day clock and alarm clock, key-repeat toggle, screen print, single keystroke ID and password transmission, color adjustment, and echo mode provide the user with a comfortable operating environment. A built-in disk command processor lets the user manage disk files directly. The PetAscii/ASCII translation tables are adjustable by the user, and can be deactivated by toggle control. Four redefinable function keys are available for storing multi-line tet strings, up to 80 characters each. Text uploads directly from disk are accommodated in either continuous or prompted mode. Automatic answerback to ENQ is provided, as well as a BREAK function for communicating with mainframes. The SMART 64 Terminal supports all direct-connect modems and most RS232 modems, including Hayes.

A separate version of the product supports the COMvoice speech synthesizer to provide the user with a TALKING terminal. The Smart 64 Talking Terminal offers all of the features of the 40 column mode in the standard package, with the exception of word-wrap control, which is replaced with two new ones more appropriate to an audio-based product: toggle voicing and phrasing of words or let-

ters. The voicing toggle allows silent, high-speed downloading of extensive tet for playback offline, or the real-time voicing of data as it is received by the modem. The phrasing control can be switched to letters for intelligible reception of unpronounceable letter groupings such as securities stock-trading symbols. Users can make back-up copies. Suggested retail for both versions is $39.95 (U.S.). Availability information from:

Microtechnic Solutions Inc.
P.O. Box 2940
New Haven, CONN 06515
203 389-8383

The FONT FACTORY VIC-1525/MPS-801 Printing Enhancement System

The FONT FACTORY will read any standard Commodore 1541 ascii sequential file, automatically format, and print out the document in any font that is selected. With this ability, the FONT FACTORY will read text files produced by many of the popular Commodore 64 word processors and produce a more presentable and interesting document. The user has full control over all page formatting, such as, page length, line width, left margin, top margin, line spacing, headers, footers, page numbering, justification, etc.

The FONT FACTORY includes an easy-to-use Font Generator to create or edit your own fonts. Fonts may be as large as 9x7 pixels, and may be printed in normal or double width formats.

The FONT FACTORY has the ability to mix up to fifteen different fonts within a single document. Thirteen embedded commands are available to give the user the ability to reformat different areas of text within the document. Use your word processor in conjunction with the FONT FACTORY to turn your 64 into a complete typesetting system!

The FONT FACTORY is user friendly and entirely menu driven. Eight preformatted fonts are provided including one with True Lower Case Descenders, when you purchase the FONT FACTORY. Additional Font Disks may be purchased separately.

Micro-W Dist. Inc.
1342B Route 23
Butler, NJ 07405
201 838-9027

CAM-64 (Call Accounting Manager) Phone Call Processing Software

Input Systems, Inc., designers and publishers of popular business software for Commodore Microcomputers, announces the release of their new software for monitoring multi-station phone usage.

CAM-64 was designed for companies with new telephone systems, or ETN systems. The program organizes SMDR data output from the phone system switching computer. CAM-64 utilizes the famous cost efficient Commodore 64 Computer, disk drive and printer. The CAM-64 System is Menu driven from an Auto Load Module, plugged into the cartridge slot of the Commodore 64. It will function on any phone system which utilizes Station Message Detail Recording (SMDR), such as Mitel, AT&T and others.

The system handles up to 2500 phone calls per disk, using a single Commodore 1541 Disk Drive. It will sort calls into several categories and sub-categories, and will send formatted printouts to a computer monitor, TV, or printer. Each format may be selected from a Menu.

CAM-64 will sort outgoing call information by:

(1) Stations/Extensions (handles up to 100 stations)
(2) Area Codes
(3) Common Carriers (up to 4, such as Microtel, Sprint, etc.)

For further information, contact:

Input Systems, Inc.
15600 Palmetto Lake Drive
Miami, FL 33157
305 253-8100

Expandable 300/1200 Baud Phone Modem With Clock/Calendar

ProModem 1200 from Prometheus Products is a Hayes compatible Bell 212A, 300 and 1200 baud phone modem with built-in clock/calendar. The unique design provides the ability to add an optional buffer memory with up to 64K of storage.

Standard features include Auto-Answer and Auto-Dial, Programmable Intelligent Dialing, Tone and Pulse Dialing, Built-in Speaker with Volume Control, separate phone and data jacks to permit switching between voice and data, and simple yet powerful diagnostics. Suggested list price is $495 (U.S.).

The ProModem 1200 can be purchased with the optional buffer installed or it can be added later. The buffer card comes with 2K of battery backed-up CMOS memory to protect time, date, operating parameter, and other data stored in memory from loss during power down. The buffer card lists for $99 (U.S.). Additional memory, in increments of 16K is available up to a maximum of 64K.

Memory in ProModem 1200 is dynamically allocated between "Directory" and "Buffer". The user can store telephone numbers, access codes, and log on messages in each directory entry. Up to 12 reference characters can be used to "call up" entries in the directory and initiate unattended dialing.

The buffer is used to store messages in the modem for transmission at a preset time, per the internal clock/calendar, to a specified group of phone numbers from the directory. In the auto-answer mode, incoming messages are automatically stored and the time recorded. Operation of ProModem can be unattended, with or without the host computer being operational.

A plug-in twelve character alphanumeric display is available for $99 (U.S.) list price. It shows operating status, diagnostic messages, phone numbers, and time/date information.

Delivery of the ProModem 1200 is 2 weeks from the factory or from stock via Prometheus' dealers. For additional information contact:

Robert Christiansen
Prometheus Products, Inc.
45277 Fremont Blvd.
Fremont, CA 94538
415 490-2370

Printer Ribbons for Commodore 1526/4023 And Others

Aspen Ribbons, Inc., of Lafayette, Colorado, is pleased to announce the Aspen Ribbons brand replacement for the Tally/Mannesmann Spirit 80 (SP80) multistrike ribbon. One hundred percent compatible to the ribbon from the original equipment manufacturer (OEM), its specifications are 1/2" x 100 ft., with multistrike film. Use this ribbon on the following printers:

- Accord DC80
- Blue Chip Enterprises M120/10
- BMC International BX-80
- Cal-Abco Legend 80
- Commodore 1526
- Commodore 4023
- Formula SP80
- ITT Xtra
- Legend 800
- Lein Yig Computer Corp. YL-80FPT
- Mitsubishi Super VCP80
- Multi-Tech Compumate CP800 Type 1
- Okidata 1600 Printer
- Ortrona AT-80
- Suminon 840
- Tally/Mannesmann Spirit 80 (SP80)
- Tally Spirit MT80 Microprinter
- Yocobushi Computer Union CO FP-80

Prices for this ribbon range from $6.00 to $3.34 (U.S.) depending on the quantity ordered. Color ribbons are not available at this time. For more information, please contact:

Aspen Ribbons, Inc.
555 Aspen Ridge Drive
Lafayette, CO 80026
303 666-5750 or 800 525-9966

The Transactor
The Tech/News Journal For Commodore Computers

PAYS
$40

per page for articles

We're also looking for professionally drawn cartoons!

Send all material to:

The Editor
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

Volume 5 Editorial Schedule

Issue#	Theme	Copy Due	Printed	Release Date
1	Graphics and Sound	Feb 1	Mar 19	April 1
2	The Transition to Machine Code	Apr 1	May 21	June 1
3	Software Protection & Piracy	Jun 1	Jul 23	August 1
4	Business and Education	Aug 1	Sep 17	October 1
5	Hardware and Peripherals	Oct 1	Nov 19	December 1
6	Programming Aids & Utilities	Dec 1	Jan 19	February 1/85

Volume 6 Editorial Schedule

1	Communications & Networking	Feb 1	Mar 21	April 1/85
2	Languages	Apr 1	May 20	June 1
3	Implementing The Sciences	Jun 1	Jul 18	August 1
4	Hardware & Software Interfacing	Aug 1	Sep 21	October 1
5	Real Life Applications	Oct 1	Nov 19	December 1

Advertisers and Authors should have material submitted no later than the 'Copy Due' date to be included with the respective issue.

100,000 CHOOSE COMAL 50,000 USERS‡

(1) DISK BASED COMAL Version 0.14

- COMAL STARTER KIT—Commodore 64™ System Disk, Tutorial Disk (interactive book), Auto Run Demo Disk, Reference Card and COMAL FROM A TO Z book.
  $29.95 plus $2 handling

(2) PROFESSIONAL COMAL Version 2.0

- Full 64K Commodore 64 Cartridge
  Twice as Powerful, Twice as Fast
  $99.95 plus $2 handling (no manual or disks)
- Deluxe Cartridge Package includes:
  COMAL HANDBOOK 2nd Edition, Graphics and Sound Book, 2 Demo Disks and the cartridge (sells for over $200 in Europe). This is what everyone is talking about.
  $128.90 plus $3 handling (USA & Canada only)

CAPTAIN COMAL™ Recommends:

The COMAL STARTER KIT is ideal for a home programmer. It has sprite and graphics control (LOGO compatible). A real bargain—$29.95 for 3 full disks and a user manual.

Serious programmers want the Deluxe Cartridge Package. For $128.90 they get the best language on any 8 bit computer (the support materials are essential due to the immense power of Professional COMAL).

ORDER NOW:

Call TOLL-FREE: 1-800-356-5324 ext 1307 VISA or MasterCard ORDERS ONLY. Questions and information must call our Info Line: 608-222-4432. All orders prepaid only—no C.O.D. Send check or money order in US Dollars to:

COMAL USERS GROUP, U.S.A., LIMITED
5501 Groveland Ter., Madison, WI 53716

TRADEMARKS: Commodore 64 of Commodore Electronics Ltd; Captain COMAL of COMAL Users Group, U.S.A., Ltd
‡ estimated

COMMODORE OWNERS

Join the world's largest, active Commodore Owners Association.

- Access to thousands of public domain programs on tape and disk for your Commodore 64, VIC 20 and PET/CBM.
- Monthly Club Magazine
- Annual Convention
- Member Bulletin Board
- Local Chapter Meetings

Send $1.00 for Program Information Catalogue.
(Free with membership).

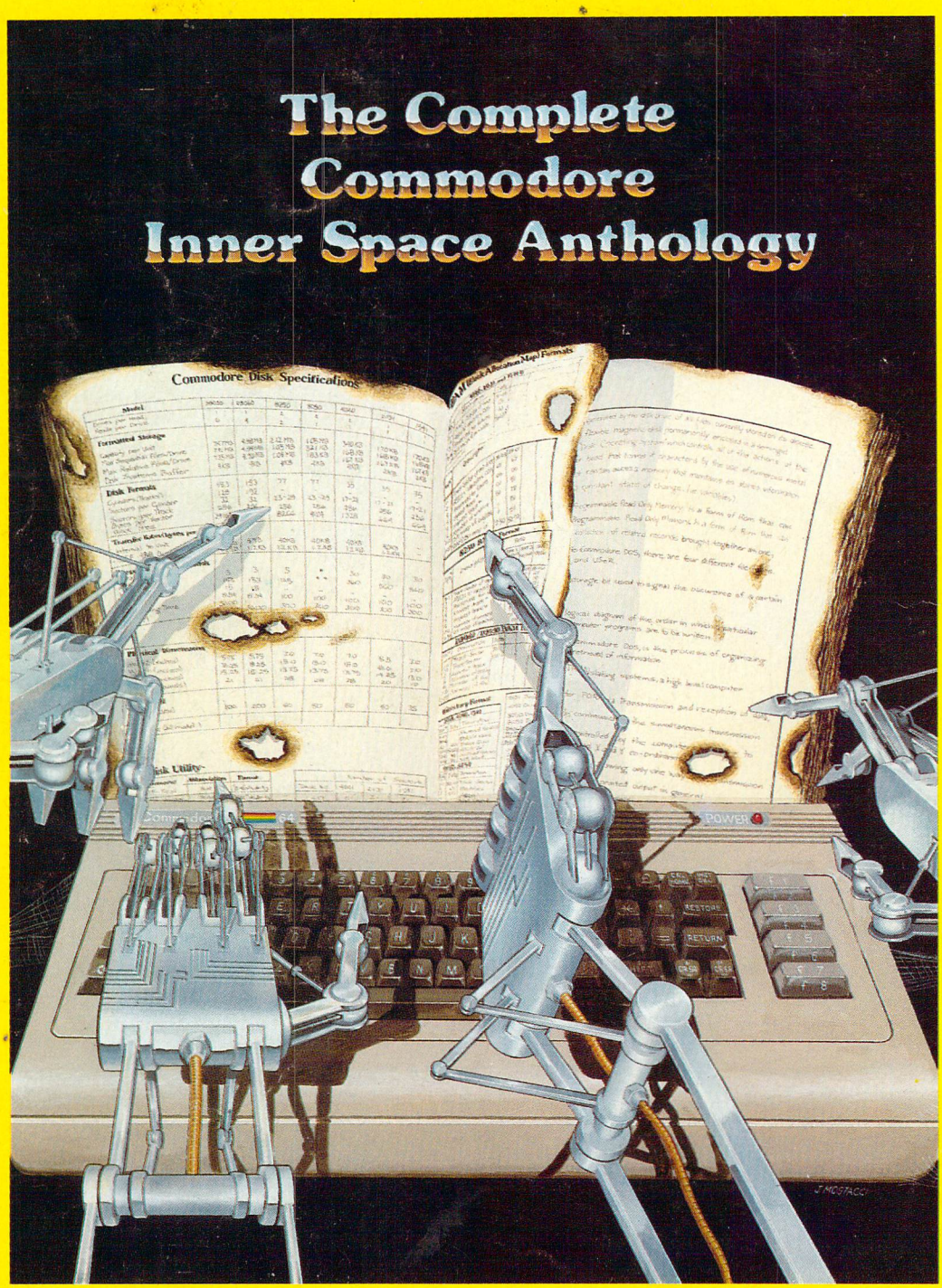
Membership Fees for 12 Months	Canada	— $20 Can.
	U.S.A.	— $20 U.S.
	Overseas	— $30 U.S.

T.P.U.G. Inc.
Department "M"
1912A Avenue Road, Suite 1
Toronto, Ontario, Canada M5M 4A1

* LET US KNOW WHICH MACHINE YOU USE *


The Transactor
Disk
Has Arrived!
Simply code your co-ordinates onto the postage powered order form and every program from each issue will be locked in, energized, and transported from our starbase directly to yours! Warp 9 will seem slow compared to the time you save typing, and the programs will give your machine that look and feel of a fresh set of Dilithium Crystals! Coast through the Neutral Zone with The Transactor Disk!
Only $7.95 Each!
6 Disk Subscription
Just $45.00!
J. Mostacci

The Complete Commodore Inner Space Anthology will look like this:



WATCH FOR IT!

January 1985