Commodore

The Transactor

comments and bulletins concerning your COMMODORE PET™

VOL 2

BULLETIN # 9

LIST in Lower Case

Robert Oei, a PET user in Mississauga, has discovered a sequence that will cause PET printers to LIST in upper/lower case rather than graphics and upper case...

The method to accomplish this is actually quite simple:

1. POKE 59468, 14
2. Enter the following line in direct mode:

```
open1,4,1:open2,4,2:print#2,"9":fora=1to
2:print#1,a;a;a;a;a;a:next:close1:close2
```

After pressing RETURN, the printer will print six "1"s and then six "2"s on the next line. If a file is then opened to the printer and the "cmd" command is given, the printer will respond with the word "ready." in lower case. Any program listing performed from then on will appear in upper/lower case. Power down and up to get graphics back.

I regret to say that I don't know what caused this phenomena to occur, except that it works. I would be most interested if you or any PET user can provide a logical reason as to why?... Robert Oei

This was tested on 2022/23 printers and found to work on some but not all.

Editor's Note

A number of readers have expressed concern about the duration of their subscription to The Transactor. Let me clear up any misunderstanding. Volume 2 Transactor subscriptions include issues 1 through 12 and are renewable upon receipt of issue #12. Therefore, anyone with a subscription to Vol. 2 should now have issues 1 to 9 and can expect 3 more bulletins before subscribing to Vol. 3 (a subscription form will appear in #12).

The Transactor is now produced on the new CBM 8032 using WordPro IV and the NEC Spinwriter.

Transactor #10

Printer Tabbing

When using TAB to print on the screen, PET looks at the current position of the cursor first (POS(0)). If the TAB argument is less than the cursors' position on the line then the data is simply printed in the spaces immediately following the last character printed. If the argument is greater than or equal to POS(0), PET subtracts POS(0) from the argument and prints the resulting number of cursor-rights.

However, when printing to the printer, the cursor is usually in column zero and TAB acts like the SPC function (the printer has no "internal cursor"). Therefore, to make TAB work on the printer, print the data to the screen first then to the printer. This can be done with duplicate PRINT and PRINT# statements or more efficiently with one "dynamic" PRINT# statement. For example:

```
10 REM OPEN OUTPUT FILES TO
     SCREEN & PRINTER
20 OPEN 3,3,1
30 OPEN 4,4,0
40 PRINT# 3+X,"ABCDEFGHIJKLMNOPQRSTUVW";
50 X=1-X : IF X THEN 40
```

Line 50 toggles X from 1 to 0 thus repeating line 40 only twice. The semi-colon is important else the POS(0) goes back to zero. When a carriage return is required on the printer the following might be inserted between PRINT# and toggle statements:

```
45 IF X THEN PRINT#4,CHR$(13);
```

Dynamic PRINT# statements are only more efficient if the DATA being printed is within quotes. If variables are used, more bytes are probably saved by duplicating the output statements.

More on Printer Output

L. D. Gardner of Leisure Books Ltd., St. Laurent Quebec, wrote in with:

Dear Sirs,

Our company uses 2 32k Commodore Pets with Centronics printers*.

The printers are used to produce formal reports of a statistical nature, and we have found that the most difficult task is not in programming the calculations, but in printing the data in acceptable format.

If 2 decimal places are required, a value of "302" is not printed as "302.00", which makes the

reports difficult to read. Also, a very small value such as ".001" is printed as "1E -03".

I have come up with a subroutine to handle this problem. Undoubtedly, it could be shortened, but for clarity is is shown in the detailed version.

I am also enclosing a simplified program which illustrates how a program can be made to update itself. It has value when not sufficient data makes tape or disk files worthwhile. At the end of the program, DATA lines are printed and the cursor is moved up. All that's neede is to hit RETURN.

SUBROUTINE TO FORMAT OUTPUT DATA

```
100 Q$=".000":LQ=LEN(Q$):LT=LOG(10)
110 P=LQ-1:IF Q$="" THEN P=0
150 INPUT"NUMBER";A
160 A1=INT(A*10^P+.5)/10^P
165 IF A1=0 THEN220
170 B1$=STR$(A1)
180 B2$=STR$(A1*10^P)
190 LG=INT(LOG(ABS(A1))/LT)
195 IF ABS(A1)=1 THEN B$="1"+Q$
200 IF ABS(A1)<1 THEN B$=LEFT$(Q$,ABS(LG))+RIGHT$(B2$,LEN(B2$)-1)
210 IF ABS(A1)>1 THEN B$=MID$(B1$,2,LG+1)+"."+RIGHT$(B2$,P)
215 IF ABS(A)=1 THEN B$="1"+Q$
220 IF A1=0 THEN B$=Q$
225 IF A1<0 THEN B$="-"+B$
230 PRINT"3"
240 PRINTA1,B$
245 PRINT"
250 GOTO 150
```

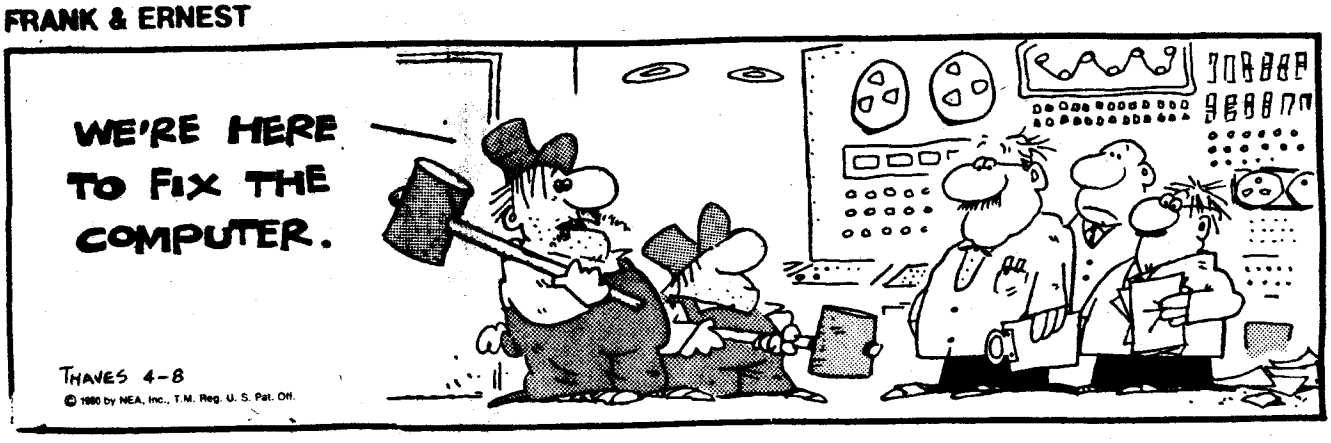
* This would also apply to Commodore printers and others

DYNAMIC TEXT PROGRAM

```
100 REM R=ROWS,C=COLUMNS
110 R=4:C=3
115 PRINT"REWIND CASSETTE THEN PUSH ANY KEY"
116 GETG$:IFG$="" THEN 116
120 DIMA(R,C)
130 FOR I=1TOR:FORJ=1TOC:READA(I,J):A(I,J)=A(I,J)*2:PRINTA(I,J):NEXT:NEXT
200 PRINT"3"
220 FORI=1TOR:PRINT2000+(10*(I-1));
230 PRINT"DATA ";
240 FOR J=1TOC
250 PRINTA(I,J);
255 IF J<C THEN PRINT"=,";
260 NEXTJ
270 PRINT
280 NEXT
285 PRINT"SAVE"
290 PRINT"
400 STOP
2000 DATA  32, 64, 96
2010 DATA  128, 160, 192
2020 DATA  224, 256, 288
2030 DATA  320, 352, 384
```

Hints on developing this program could be found in the Disk Merge article (Transactor 8) and Paul Barnes article on finding DATA statements (this issue).

Consider a program that could receive data from the keyboard and then SAVE itself with the data included...dynamically! Such a program would have to first determine a line number for the DATA statement. This would be PRINTed followed by "DATA" and your information. By POKing the keyboard buffer (623-632 see also 158 # characters in buffer) and ENDing program execution, carriage returns could be released over top of the line upon relinquishing control to PET. The screen editor would enter the DATA statement into text and a second carriage return could be placed over a RUN or GOTO xxx back into the program. Once the program reaches memory capacity, it can be filed for future reference and a new program started. Ideal as a light data-base.



W.T. Garbutt
Mississauga, Ont.

Waterloo Extended PET BASIC

Waterloo University recently announced the availability of an extended Basic for the Commodore 16/32k PET. The 2k ROM chip resides in expansion ROM socket at 9000, comes with a 160 page user manual and costs $75 in single quantities. Waterloo University is introducing the product in a series of 3-week seminars of particular interest to high school teachers involved in the programming sciences. The seminar participant attends a one day introductory session, takes home a PET and returns for a 2 day concluding session. Yours truly is presently attending the course and will file an in depth report of the Basic and the course in a future Transactor. Essentially the new Basic supports all existing Pet Basic syntax while adding powerful commands such as LOOP...ENDLOOP...WHILE...ENDIF...CALL...PROC(edure),etc. to permit a fully structured program approach consistent with accepted industry practices. For further information contact;

Prof. W. J. Graham,
Computer Systems Group,
University of Waterloo,
Waterloo, Ontario Canada,
N2L 3G1,
(519) 885-1211

Coming Reviews or Articles

KRAM 2.0

United Software of America's 2k ROM provides the heart of an advanced keyed Random Access (disk) management system, essential for comprehensive data base management implementations.

Rabbit

Eastern House Software's (Carl Moser) utility software that among other things increases PET tape speed by a factor of four.

ROM Expansion

What to do about conflicting expansion ROM assignments. How to use the expansion Rom area of the PET.

Structured Symbolic Basic Compiler

Softside Software's Basic program that implements a structured two pass compiler on the PET!

Computer Music

Beginning in the next issue of The Transactor a definitive series of articles on all you ever wanted to know about PET and music.

Micro-Computer Communication

A look at MicroNet and the Source, as well as the where's, why's and how's of establishing contact with other micro-computer users over telephone, radio etc.

New Products

Scheduled for imminent release are several superb software packages.....Personal Software's VISICALC data base management with advanced mathematical modelling features..... SARGON super chess for the PET....an 80 character video display equipped PET.... a 1.1 megabyte dual floppy 5 1/4" disk PET peripheral.

Meetings or Events

12th June

The 1st annual Sheridan College Music Symposium: A one day seminar featuring an expert look at the state of the art Computer Music hardware and software technologies including lectures by Hal Chamberlin, Jim Butterfield, Prof. Sterling Beckwith and others. For further information contact;

Margo Martin,
1430 Trafalgar Rd.,
Oakville, Ontario,
L6H 2L1,
(416) 845-9430 ext 132.

13th June

Tommorrow is Here: The Impact of the New Technogy On Education. A one day International Conference of interest to Educators from Pre-school to Post Secondary levels under the auspices of the Hamilton Board of Education, the Ontario Mathematics Coordinators' Association and McMaster University. For further information contact;

TOMMORROW IS HERE,
P.O. Box 558,
Hamilton, Ontario,
L8N 3L1

15th-18th June

The 1980 Summer CES (Consumer Electronics Showplace)

McCormack Place, Chicago, Il. The definitive dealer and trade show that covers the whole consumer electronics ball of wax... audio, video, communication and computers.

Special Note:

The Transactor is a national periodical. As such we would like to keep readers abreast of major events or meetings of regional or local interest. If you are aware of such occurances please drop a note to The Transactor's editor at the following address;

Editor,
The Transactor,
Commodore Business Machines Ltd.,
3370 Pharmacy Ave.,
Agincourt, Ontario,
M1W 2K4
(416) 499-4292

Please provide as much lead time as possible (2 to 3 months is a safe bet.)

Interested in Micro-computer Networking

PET Community members interested in forming computer bulletin board communication networks are asked to contact the Editor. The Transactor will regularly publish articles and software, act as a clearing centre or catalyst to further the establishment of this important facility.

FLASH! FLASH! FLASH!

Educators....Commodore Canada is happy to announce that the three for two Pet offer is back. For a limited time.... to August 15, 1980 bonafide educational institutes can receive, from Commodore, one free Pet for every two they purchase from an authorized Commodore dealer. For further details contact your local dealer or the following address;

Commodore Business Machines Ltd.,
3370 Pharmacy Ave.,
Agincourt , Ontario,
M1W 2K4,
(416) 499-4292

Attention Programmers!

Mr. Bill Twyman has recently been apointed Software Manager for Commodore Canada. To further develop the software department of the computer division, Bill has submitted the following:

Dear Transactor Reader

Many things - exciting things are happening here at Commodore.

As all of you know, a PET won't do very much without software. As a result, we are soliciting saleable software which can be marketed by Commodore on a national basis. All you have to do is write it and we will handle the rest. No matter how specific the use might be, there are other PET users who would be interested. This is also a good way to make the PET pay for itself.

Simply do the following: send me a working copy of the program plus written documentation. All submissions will be treated with the utmost confidentiality and will returned to you after evaluation.

Any questions will gladly be answered. Please call me at:

Commodore Business Machines
416 499 4292

Let's see what's out there!

Bill Twyman
Software Manager

Output to the screen is quite straightforward. Load the ASCII character into the A register; then call the routine at FFD2. Special characters, such as cursor movements, will be honoured in the usual way.

The GET activity gives no trouble, either, except for one minor situation. To do a GET, call FFE4 and the character will appear in the A register. If you don't have a character available, the subroutine will return zero in the A register. Since you can't get an ASCII zero from the keyboard, recognize this as a "no-character" situation and arrange to deal with it as desired.

INPUT is a little trickier. When you call FFCF for Input, you'll get one character back. This seems like a GET, but it's really quite different. The first time you call, it will prompt and get an input, transfering it via the screen in the usual way; then it will edit out leading and trailing spaces and quote marks. After doing all this work, it will deliver the first character to you. On subsequent calls, it will deliver following characters. When it has delivered the whole input, it will deliver a Return character to signal you've got it all. After that, it starts over.

Beginners will be happier using the GET call.

Peripheral Input/Output

Surprisingly easy, once you have the above techniques mastered.

Start by OPENing the file in BASIC, before you go to machine language. When you're ready to the actual activity, the machine language sequence is as follows:

Load X with the logical file number;
For INPUT or GET, call FFC6 to set the input channel;
For output, call FFC9 to set the output channel;

Now use you INPUT, GET, or output calls as described above;

Finally, restore the normal input/output channels with a call to FFCC. Careful! This routine changes the A register to zero.

Wind up your program in BASIC by closing all files, as usual.

When you're INPUTting or GETting from an external device, keep an eye on the status word, ST (located at 020C in original ROM, or at 96 in 2.0 ROM). It will warn you when you reach the end of a input file.

The above procedure isn't too hard, and it's likely to carry through to newer versions of ROM when they appear.

Have you ever wanted to program something like...

MID$ (A$, 10) = "Name, Address, etc..."

...well now you can!...thanks to another fabulous routine by Bill Maclean of BMB CompuScience, Milton Ontario. The routine works only with PETs using the 2.0 ROM set.

This is a little utility to allow a programmmer to change a substring within a main string. Its primary uses are manipulating data records in disk files and setting up formatted printer or screen outputs. It is called with the following command

SYS 826,A$,B$,X

This command string will cause the string A$ to be placed within string B$, starting at the 8th character. The A$,B$,and X are all variables. Any variables can be used. The programmer is responsible for assuring that the length of the main string is not exceeded.

The machine language routine can be entered using the resident monitor and cursor editing the screen display. The code is completely relocatable and can be placed anywhere or relocated anywhere. The calling address (826 above) should be the address of the first byte of the program.

```
     PC  IRQ  SR AC XR YR SP
.;  0005 E62E 30 00 5E 04 F5
.M 033A 0382
.:  033A 20 F8 CD 20 9F CC A0 00
.:  0342 B1 44 85 00 C8 B1 44 85
.:  034A 01 C8 B1 44 85 02 20 F8
.:  0352 CD 20 9F CC A0 01 B1 44
.:  035A 85 0F C8 B1 44 85 10 20
.:  0362 F8 CD 20 9F CC 20 D2 D6
.:  036A A5 12 F0 03 4C 03 CE C6
.:  0372 11 A5 0F 18 65 11 85 0F
.:  037A 90 02 E6 10 A0 00 B1 01
.:  0382 91 0F C8 C4 00 D0 F7 60
```

If you'd like to see what's going on on the GPIB - and if you can borrow an extra PET and IEEE interface cable - this program will help.

It shows the current status of four of the GPIB control lines, plus a log of the last nine characters transmitted on the bus.

The four control lines are NRFD, NDAC, DAV and EOI. It would be nice to show ATN too, but I couldn't fit this in: it's detected in a rather odd way in the PET so that fitting it in is somewhat too tricky for this simple program.

The last nine characters are shown in "screen format". This means that you'll have to do a little translation work to sort out what some of them mean. On the other hand, it allows you to see characters that otherwise wouldn't be printed. A carriage return, for example, shows up as a lower case m; this is a little confusing at the start, but you'll quickly get used to it and it's handy to see everything that goes through. Don't forget that original model PETs may show upper and lower case reversed.

I had hoped to show which characters were accompanied by the EOI signal. It turned out that time is critical - the bus works very fast - and that adding this feature would cut down the number of displayed characters from nine to five. I opted for the bigger count and dropped the EOI log feature.

The high speed of the bus makes it difficult to watch the control lines in real time. When the "active" PET is exchanging information with disk or printer, everything is happening very fast, and the "logic analyzer" PET will show an amazing flurry of activity on the control lines. Only when the activity stops or hangs up will you be able to see the lines in their static conditions.

You may use the program to chase down real GPIB problems, or just to gain insight on how the bus works. Either way, it will come in handy if you can borrow that extra PET unit.

```
10 REM IEEE WATCH    JIM BUTTERFIELD
20 REM  MAY 1980
30 POKE59468,14:PRINT"□ DAV NRFD NDAC  EOI":PRINT"  ↑    ↑    ↑    ↑■"
40 PRINT"=123456789=■"
50 SYS1200
```

```
                        ; IEEE WATCH   JIM BUTTERFIELD  MAY/80
110:    04B0                    *=    $4B0
120:    04B0            DFLAG   =     $B1
130:    04B0            DNNSAV  =     $B2
140:    04B0            EOISAV  =     $B3
200:    04B0 46 B1      START   LSR   DFLAG
210:    04B2 78                 SEI
220:    04B3 AD 12 E8   MAIN    LDA   $E812
230:    04B6 C9 EF              CMP   #$EF
240:    04B8 D0 02              BNE   CONT
250:    04BA 58                 CLI
250:    04BB 60                 RTS
280:    04BC AC 10 E8   CONT    LDY   $E810         ;EOI
290:    04BF AD 40 E8           LDA   $E840         ;DAV, NRFD, NDAC
300:    04C2 AE 20 E8           LDX   $E820         ;DATA
310:    04C5 29 C1              AND   #$C1          ;EXTRACT BITS
320:    04C7 C5 B2              CMP   DNNSAV
330:    04C9 D0 11              BNE   DNN
340:    04CB 98                 TYA
350:    04CC 29 40              AND   #$40          ;EXTRACT EOI
360:    04CE 0A                 ASL   A
370:    04CF 49 A0              EOR   #$A0
380:    04D1 C5 B3      EOI     CMP   EOISAV
390:    04D3 F0 DE              BEQ   MAIN
400:    04D5 85 B3              STA   EOISAV
410:    04D7 8D 61 80           STA   $8061
420:    04DA D0 D7              BNE   MAIN
                        ;ACTIVITY SEEN - UPDATE SCREEN
430:    04DC 85 B2      DNN     STA   DNNSAV
440:    04DE 29 80              AND   #$80
450:    04E0 49 A0              EOR   #$A0
460:    04E2 8D 52 80           STA   $8052
470:    04E5 10 1D              BPL   NDAV          ;NO DAV SEEN
500:    04E7 A4 B1              LDY   DFLAG
510:    04E9 30 1B              BMI   DCONT         ;DAV SEEN BEFORE
520:    04EB 85 B1              STA   DFLAG
530:    04ED 85 B2              STA   DNNSAV
540:    04EF A0 00              LDY   #0
550:    04F1 B9 A2 80   SCROL   LDA   $80A2,Y
560:    04F4 99 A1 80           STA   $80A1,Y
570:    04F7 C8                 INY
580:    04F8 C0 08              CPY   #8
590:    04FA D0 F5              BNE   SCROL
600:    04FC 8A                 TXA
600:    04FD 49 FF              EOR   #$FF
600:    04FF 8D A9 80           STA   $80A9
610:    0502 B0 AF              BCS   MAIN
640:    0504 85 B1      NDAV    STA   DFLAG
650:    0506 A5 B2      DCONT   LDA   DNNSAV
660:    0508 29 40              AND   #$40          ;NRFD
670:    050A 0A                 ASL   A
680:    050B 49 A0              EOR   #$A0
690:    050D 8D 57 80           STA   $8057
700:    0510 A5 B2              LDA   DNNSAV
710:    0512 29 01              AND   #$1           ;NDAC
720:    0514 4A                 LSR   A
730:    0515 6A                 ROR   A
740:    0516 49 A0              EOR   #$A0
750:    0518 8D 5C 80           STA   $805C
760:    051B D0 96              BNE   MAIN
```

COMPUTER PHILOSOPHY

W.T. Garbutt
Mississauga, Ont.

> "It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light , it was the winter of despair, we had everything before us, we had nothing before us, we were all going directly to Heaven, we were all going directly the other way--in short, the period was so much like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only."

So began Charles Dickens in the famous introduction of the 18th Century setting for "A Tale of Two Cities" over a century ago! As with all great literature, the vision Dickens gives is perhaps centuries ahead of its time, in any event just as germane today as it was when Dickens first wrote it. Today Western civilization is on a precipice; the brink of a great leap forward for Humanity or a momentary or perhaps fatal fall.

While less than one-sixth of the world's population consumes over 80% of world resources, hundreds of millions fight daily to survive on the remainder. And those who survive remember!

For the fortunate few, the leap forward has begun. While scholars will argue for generations as to the precise time, (there probably is not one) the race to the Moon will likely symbolize the start for many. None of the generation of the sixties will forget President Kennedy's proclamation to reach the Moon by the end of that decade (nor will they likely forget his tragic and senseless death). The last brief decade has witnessed a massive expansion of knowledge, the extent of which is beyond human comprehension. If all knowledge known were increased by the order of a magnitude in the 70's, is there any doubt the increase will be two orders of magnitude this decade? Is there any certainty that this exponential knowledge explosion will quench humankinds' thirst by the 21st century?

Yes. There is serious doubt if the knowledge does not bring with it a maturity, a compassion for fellow man, a sharing and a sacrifice . The dismal scenario has already been summarized by Mr. Dickens in the 18th century: war, pestilence, famine and destruction. And even with our present limited knowledge is there any doubt that a similar order of a magnitude, an exponential order of horror could soon be our legacy.

There are disquieting signs that the future optimism, our traditional western belief in historical progress may be open to serious question. What Western civilization must guard against is introspection. Cultural isolation will almost certainly provide the impetus for an apocalypse. This

introspection can take many forms. Most familiar to the readers of The Transactor is the discipline of electronics and more particularly computers.

Certainly this is a paradox. The computer, the single most important element in the recent knowledge explosion, the genie that could provide the key to the crucial inter-relationships in what at present is an incomprehensible mass of data, surely could not be viewed as retrenchment. On the contrary if a limited number of well educated and affluent people become preoccupied with the computer solely for their own personal intellectual gratification that is precisely what could happen.

However it need not happen. What is required is an atmosphere of freedom, freedom of thought, expression, knowledge freely accessible to all as well as freedom from hunger and poverty. A vast educational challenge exists certainly as great as that witnessed in the race to the Moon. And the micro-computer is ideally suited as a central instrument in meeting the challenge.

One of the major micro-computer applications will be communications. The micro-computer must become as universally used as the light bulb. It must be available to the 'outbacks' of civilization as well as the present temples of Western civilizations (Universities). Computer literacy must become, with literacy, the prime objective of man after the basic survival improvements over hunger and poverty. No longer must language create communication barriers between nations, no longer may state bureaucracies hide knowledge clandestinely or inadvertently, no longer can corporations shape consumer habits through controlled ignorance.

These freedoms of course carry risks. Change creates instability. Governments and citizens fear unbridled change especially anarchy. But the risks of not accepting the challenge is greater. We could lose all the significant advances that civilization to date has witnessed. The present world population could turn away from a Western tradition that failed to solve humankinds most immediate problems.

The computer can become a central tool to manage today's knowledge, freeing man's creative and underutilized imagination, permitting him time to examine himself.

Today Western civilization senses the future. One has only to witness the long lineups and multi-billion dollar box office receipts for 'science fiction' movies. Today's audiences are seeing implemented yesterday's dreams. And super realistic depictions of tomorrow. The audiences understand, as never before, the revolutionary technological concepts. Indeed they will accept nothing less. In many instances these special effects were only made possible with the aid of the computer. We must not permit this phenomenon to become a form of escapism. It is important to feed the scientific needs of our citizens. But we must also consider the equally important spiritual needs. eg. beliefs and values.

The computer can provide assistance to man. For every self-gratifying use there is equally a use that will help meet the challenges. From simple energy conservation methods such as controlling house temperature, or arranging car pools to complex techniques to improve work effiency computer uses abound to cut waste.

A recent industry census indicated there are over 500,000 computers in personal use today. Within two years personal ownership is projected to quintuple. Think of the immense number of applications such a user population can generate.

As a member of the micro-computer community, ask yourself the question "What uses can I derive and implement that will aid, no matter how insignificant it may appear, in meeting the challenges of today?" Talk to fellow users , share your ideas and you will likely find yourself in the vanguard of tommorrow's leaders.

A BOOK REVIEW

Tom Wojdylo
SES (Computing) Inc.

About a month ago I received a copy of "PET and the IEEE 488 Bus (GPIB)" by Eugene Fisher and C. W. Jensen. I took an immediate liking to it because of the very readable approach the authors take to a potentially very dry technical subject. The use of diagrams, cartoons and timing sequences also enhances the reader's understanding of what is being discussed.

Chapter one discusses systems and the developement of the IEEE STD 488 Bus.

Chapter two describes each line of the GPIB, the basic bus structure, control lines and data bus lines. A handshake procedure with the PET as controller/talker and controller/listener is detailed.

The fourth and fifth chapters get into programming the PET for the IEEE 488 Bus. There is extensive coverage of the BASIC statements and commands and how they are executed, with detailed diagrams of the timing sequences which are generated on the GPIB Bus.

In the last three chapters, typical interfacing applications to standard and non-standard devices are explained. For example, there is information on printers which are usually non-standard devices and an HP Digital Voltmeter which has an IEEE STD 488 Bus.

All the data in the book is related to both old and new ROMs and this is a very important feature.

I found this book addresses PET users at all levels - from the hobbiest to the digital circuit designer; from the college student to the scientist.

It is an excellent reference for community college instructors and students which help them both to understand the fundamentals of the IEEE 488 Bus.

In today's rapidly changing technology, a book with such exaustive, up-to-date information is an excellent reference worth having in any PET user's library.

CROSS - REFERENCE

Jim Butterfield, Toronto
also in Compute #4

One of the handy things about the 2040 disk system is that it allows you to read programs - or write them, for that matter - as if they were data files.

The possibilities are endless: you can analyze or cross-reference programs; renumber them; repack them into minimum number of lines deleting spaces, comments, etc.; or even create a program-writing program that is tailor-made for a particular job.

This program does cross-referencing of a BASIC program. It's written in BASIC: that means that it won't run too fast (all those GET statements) but you can read what it's doing fairly easily.

There are two types of cross-references normally needed for a BASIC program. One is the variable cross-reference: where do I use B$? The other is a line-number cross-reference: when do I go to line 360 ? CROSS-REF does either. An example of both types is shown - the program in this case did the cross-references of itself.

CROSS REFERENCE - PROGRAM CROSS-REF

```
A       270   280   300   310   390   400
A$      180   240   260   270   300   460   490   500   510   520   560
        570   580
A$(     100   200   330   340   350   360
B       190   200   220   320   330   340   350   360
B$      180   240   480   500   520   580   590
B$(     100   120   480
C       280   370   390   410   420   430   440   450   540   550
C$      480   520   580   620
C(      100   140   150   160   310
C1      280   310   370   420   440
C2      130   150   205   280   370   380   450   565
C9      310   410   420   440   470   480
J       140   150   200   210   220   330   340   350   560   630
K       200   210   220   320   340   350   360   565   570   580
L       260   280
L$      200   206   280
M$      330   340   360   370   450   460
P$      170   550
Q$      120   510
S$      120   280   590   610   620
X       200   220   560
X$      200   205   206   210   220   560   580   590
X$(     100   210   220   560
Y       590   600   610
Z       540   600
Z$      130   530   540
```

```
100 DIM A$(15),B$(3),X$(500),C(255)
110 PRINT"CROSS-REF    JIM BUTTERFIELD"
120 Q$=CHR$(34):S$="      ":B$(1)=Q$:B$(3)=CHR$(58)
130 INPUT"VARIABLES OR LINES";Z$:C2=5:IFASC(Z$)=76THENC2=6
140 FORJ=1TO255:C(J)=4:NEXTJ:FORJ=48TO57:C(J)=6:NEXTJ
150 IFC2=5THENFORJ=65TO90:C(J)=5:NEXTJ:FORJ=36TO38:C(J)=7:NEXTJ:C(40)=8
160 C(34)=1:C(143)=2:C(131)=3
170 INPUT"PROGRAM NAME";P$:OPEN1,8,3,"0:"+P$+",P,R"
180 GET#1,A$,B$:IFASC(B$)<>4THENCLOSE1:STOP
190 IFB=0GOTO240
200 PRINTL$;:K=X:FORJ=BTO1STEP-1:PRINT" ";A$(J);:X$=A$(J)
205 IFC2=6ANDLEN(X$)<5THENX$=" "+X$:GOTO205
206 X$=X$+L$
210 IFX$(K)>=X$THENX$(K+J)=X$(K):K=K-1:GOTO210
220 X$(K+J)=X$:NEXTJ:X=X+B:PRINT:B=0
230 REM: GET NEXT LINE, TEST END
240 GET#1,A$,B$:IFLEN(A$)+LEN(B$)=0GOTO530
250 REM GET LINE NUMBER
260 GET#1,A$:L=LEN(A$):IFL=1THENL=ASC(A$)
270 GET#1,A$:A=LEN(A$):IFA=1THENA=ASC(A$)
280 C=C2:C1=-1:L=A*256+L:L$=STR$(L):IFLEN(L$)<6THENL$=LEFT$(S$,6-LEN(L$))+L$
290 REM GET BASIC STUFF
300 GET#1,A$:A=LEN(A$):IFA=1THENA=ASC(A$)
310 C9=C(A):IFC9>C1GOTO380
320 K=0:IFB=0GOTO360
330 FORJ=1TOB:IFA$(J)=M$GOTO370
340 IFA$(J)<M$THENNEXTJ:K=B:GOTO360
350 FORK=BTOJSTEP-1:A$(K+1)=A$(K):NEXTK
360 B=B+1:A$(K+1)=M$
370 C=C2:C1=-1:M$=""
380 IFC2=5GOTO420
390 IFA=137ORA=138ORA=141ORA=167THENC=6:GOTO470
400 IFA=44ORA=32GOTO470
410 IFC9<>6THENC=9:GOTO470
420 IFC9=CTHENC=-1:C1=4
430 IFC>6GOTO470
440 IFC<0ANDC9>C1ANDC9>6THENC1=C9:GOTO460
450 IFC2=5THENIFLEN(M$)>2ORC>0GOTO470
460 M$=M$+A$
470 ONC9+1GOTO190,480,480,480:GOTO300
480 B$=B$(C9):C$=""
490 GET#1,A$:IFA$=""GOTO190
500 IFA$=B$GOTO300
510 IFA$<>Q$GOTO490
520 A$=B$:B$=C$:C$=A$:GOTO490
530 CLOSE1:INPUT"PRINTER";Z$
540 C=3:Z=6:IFASC(Z$)=89THENC=4:Z=12
550 OPEN2,C:PRINT#2:PRINT#2,"CROSS REFERENCE - PROGRAM ";P$
560 X$="":FORJ=1TOX:A$=X$(J)
565 IFC2=6THENK=6:GOTO580
570 FORK=1TOLEN(A$):IFMID$(A$,K,1)<>" "THENNEXTK:STOP
580 B$=LEFT$(A$,K-1):C$=MID$(A$,K+1):IFX$=B$GOTO600
590 PRINT#2:Y=0:X$=B$:PRINT#2,X$;LEFT$(S$,5-LEN(X$));
600 Y=Y+1:IFY<ZGOTO620
610 Y=1:PRINT#2:PRINT#2,S$;
620 PRINT#2,LEFT$(S$,6-LEN(C$));C$;
630 NEXTJ:PRINT#2:CLOSE2
```

190	470	490			
205	205				
210	210				
240	190				
300	470	500			
360	320	340			
370	330				
380	310				
420	380				
460	440				
470	390	400	410	430	450
480	470				
490	510	520			
530	240				
580	565				
600	580				
620	600				

Reading a BASIC Program as a File

To read a BASIC program, you must OPEN it as a file, using type P for PRG rather than S for SEQ. Line 170 of CROSS-REF does this.

If you read a zero character from the program (that's CHR$(0), not ASCII zero which has a binary value of 48), the GET# command gives you a small problem: it will give you a null string instead of the CHR$(0) you might normally expect. You need to watch this condition and correct it where necessary: you'll see this type of coding in lines 260, 270 and 300.

The first thing to do when you OPEN the file is to get the first two bytes. These represent the program start address, and should be CHR$(1) and CHR$(4) for a normal BASIC program starting at hexadecimal 0401 (see line 180).

Now you're ready to start work on a line of BASIC. The first two bytes are the forward chain. If they are both zero (null string) we have reached the end of the BASIC program; otherwise, we don't need them for this job (see line 240).

Continuing on the BASIC line: the next pair of bytes represent the line number, coded in binary. We're likely to need this, so we calculate it as L (lines 260 and 280) and also create it's string equivalent, L$. We take an extra moment to right-justify the string by putting spaces at the front so that it will sort into proper numeric order.

From this point on we are looking at the text of the BASIC line until we reach a zero which flags end-of-line. At that time we go back and grab the next line.

Detailed Syntax Analysis

When digging out variables or line numbers, we have several jobs to do. As we look through the BASIC text, we must find out where the variable or line number starts. For a variable, that's an alphabetic character; for a line number, it's the preceeding keyword GOTO, GOSUB, THEN or RUN followed by an ASCII numeric.

Once we've "aquired" the variable or line number, we must pick up its following characters and tack them on. For line numbers it's strictly numeric digits. For variables, things are more complex. Both alphabetic and numeric digits are allowed, but we should throw away all after the first two since GRUMP and GROAN are the same variable (GR) in PET BASIC. We must also pick up a type identifier - % for integer variables or $ for strings - if present. Finally, we have to spot the left bracket that tells us we have an array variable.

To help us do this rather complex job, we construct a character type table. Each entry in the table represents an ASCII character, and classifies it according to its type. Numeric characters are type 6. If we're looking for variables, alphabetic characters are type 5, identifiers are type 7, and the left bracket is type 8.

To help us in scanning the BASIC line, we define the end-of-line character as type 0; the quotation mark as type 2; the REM token as type 3; and the DATA token as type 4.

Every time we get a new character from BASIC, we get its type from table C as variable C9. If we're looking for a new variable or line number, we see if it matches C - alphabetic for variables, numeric for line numbers. Once we find the new item, we kick C out of range and start searching based on the value of C1. This mechanism means that we can search for a variable starting with an alphabetic, and then allow the variable to continue with alphabetics, numerics or whatever.

To summarize variables in this area: A is the identity of the character we have obtained from the BASIC program, and C9 is its type. If we're searching, C is the type we are looking for; otherwise it's kicked out of range, to -1 or 9. C1 tells us we're collecting characters and what type we're allowed to collect. C2 is out variables/line numbers flag; it tells us what we're looking for. M$ is the string we've assembled.

The routine from 480 to 520 scans ahead to skip over strings in quotes and DATA and REM statements.

Collecting the Results

For each line of the BASIC program we are analyzing, we collect and sort any items we find, eliminating duplicates. They are staged in array A$ in lines 320 to 370.

When we're ready to start a new line, we add this table to our main results table, array X$, in lines 200 to 220. To

save sorting time, we merge these pre-sorted values into the main table. At this point, our data has the line number stuck on the end; this way, we're handling two values within a single array.

Because the merging of the two tables must start at the top so that we can make room for the new items, the items are handled in reverse alphabetic order. We print this to the screen so that you can watch things working. At BASIC speed, this program can take quite a while to run; it's nice to confirm that the computer is doing something during this period.

Final Output

We finish the job starting at line 530. It's mostly a question of breaking the stuck-together strings apart again and then checking to see if we need to start a new line.

Do Your Own Thing

The size of array X$ determines how large a program you can handle. The given value of 500 is about right for 16K machines; with 32K you can raise it to 1500 or so.

If you're squeezed for space, change array C to an integer array C%. As you can see from the cross reference listing, you'll need to change lines 100, 140, 150, 160 and 310 - see how handy the program is ?

As mentioned before, run time is slow. A machine language version - or even a BASIC program with machine language inserts - would speed things up dramatically.

NOTE: Some ASCII printers may give double spaced output. If this is a problem the PRINT#2 statements in 590 and 610 should be changed to PRINT#2,CHR$(13);.

Better Auto Repeat

David Berezowski of ASCII Computing, Thunder Bay Ontario, has submitted another repeat key program which might be used instead of the one printed in Transactor #7.

```
0 REM RELOCATABLE AUTO-REPEAT BY...
1 REM DAVID BEREZOWSKI
2 REM ORIGINAL CODE TAKEN FROM BEST OF TRANS. VOL 1
3 REM UPDATED FOR NEW ROM AND PUT INTO RELOCATABLE FORM BY DAVID BEREZOWSKI
4 REM RELOCATABLE FORMAT TAKEN FROM J. BUTTERFIELDS TRACE ROUTINE
5 K=0
10 PRINT"■■THIS PROGRAM LOCATES ■AUTO-REPEAT■ IN"
20 PRINT"ANY SIZE MEMORY THAT IS FITTED...■"
30 IFPEEK(65000)=254THENE=52:D=0:GOTO60
40 IFPEEK(65000)<>192THENPRINT"?? I DON'T KNOW YOUR ROM ??":END
50 E=134:D=4:K=3:FORJ=1TO56:READZ:NEXTJ
60 PRINT"I SEE THAT YOU HAVE AN ";
70 IFE=134THENPRINT"ORIGINAL";
80 IFE=52THENPRINT"UPGRADE";
90 PRINT"  R O M."
95 FORZ=1TO2000:NEXT
100 DATA 162,3,181,255,157,45,3,202,208
101 DATA 248,56,169,233,229,145,133,145
102 DATA 96,165,166,201,255,208,10,169
103 DATA 0,133,15,169,48,133,16,208,19
104 DATA 230,15,165,16,197,15,176,11
105 DATA 169,6,133,16,162,255,134,151
106 DATA 232,134,15,76,46,230
150 REM**************************
160 REM*  END OF UPGRADE DATA *
170 REM**************************
200 DATA 162,3,181,255,157,132,3,202,208
201 DATA 248,56,169,233,237,26,2,141,26,2
202 DATA 96,173,35,2,201,255,208,10,169
203 DATA 0,133,60,169,48,133,61,208,20
204 DATA 230,60,165,61,197,60,176,12
205 DATA 169,6,133,61,162,255,141,3,2
206 DATA 232,134,60,76,133,230
1000 S2=PEEK(E)+PEEK(E+1)*256
1005 S1=S2-56-D
1010 FORJ=S1TOS2-1
1020 READX:POKEJ,X:NEXTJ
1030 S=INT(S1/256):T=S1-S*256
1040 POKE0,76:POKE1,T+18+D/2:POKE2,S
1050 POKEE,T:POKEE+1,S
1060 POKEE-4,T:POKEE-3,S
1130 PRINT"■■===AUTO-REPEAT==="
1140 PRINT"■TO ENABLE: SYS"S1
1150 PRINT"TO DISABLE: SYS"S1
1160 PRINT"■CHANGE SPEED WITH: POKE"S1+43+K"■,X
1170 PRINT"CHANGE DELAY WITH: POKE"S1+29+K"■,X (X>10)
1180 PRINT"■■TO EXPERIENCE THE FRUSTRATION FROM"
1190 PRINT"■KEY-BOUNCE THAT ALL TRASH-80 OWNERS"
1200 PRINT"■MUST PUT UP WITH, TRY POKING "S1+29+K
1210 PRINT"■WITH VALUES LESS THAN 5!"
1220 PRINT"■NOTE:YOU MUST DISABLE AUTO-REPEAT"
1230 PRINT"BEFORE USING THE CASSETTE"
```

THE 'UNWEDGE' -- A TAPE APPEND AND RENUMBER PROGRAM

Many PET users have only tape input for their system. (Until a few weeks ago, I was among that majority). If you have often wanted to append a favourite subroutine, you may have been stymied. I use the input subroutine from Cursor #4 ('INP', Oct 1978) in many programs. You may have your own favourites, if only there was an easy way to append them.

Another possibility might entail merging several short programs together, such as the Osborne & Associates: 'Some Common Basic Programs'. Individually, most are quite short, which could permit a 'menu-driven' composite program.

I am sure that other applications may come to mind.

Most of us have a Renumber program of some sort, whether it is in Machine Language, or as a Basic subroutine which is tacked on to the end. The main disadvantage is that the location in memory is fixed.

Welcome to the Upgrade-ROM combination of the two, which offers:

***Machine Language for speed of operation.
***Full relocatability to anywhere in RAM. When run, 'APPEND/RENUM' places itself in high memory and protects itself from intrusion by Basic.
***The 'Unwedge' (or '<' key) attaches itself to the operating system. It may be used in the Direct Mode with a minimum of user input.
***It is compatible with the Disk Wedge routine. Although the Append function works only with tape input, both Wedges may be active simultaneously.
***The routine may be implemented and de-activated with the same 'SYS' command.
***Uses only 771 bytes of memory.

Bill Seiler deserves the credit for nearly all the coding. In Pet User Notes (Vol I,#7 Nov/Dec 1978), he presented his 'PET Renumber 3.0' for Original ROM. This has been converted to Upgrade ROM and allows for user input of parameters.

In our own Transactor (Vol 2,#3 July 1979), Bill presented the Append Wedge for Original ROM. In the same Pet User Notes issue, Jim Russo and Henry Chow gave us M7171. This is a high-monitor with merge capability for Original ROM. Features of both of these were converted to Upgrade ROM, and several additions made.

TO USE THE PROGRAM:

1. Copy the program listing.
2. Save it immediately, since Machine Language routines have a nasty habit of crashing, with even the slightest error.
3. 'RUN' the program, and copy down the information which appears (after about 20 seconds).

4. Save the Machine Language code with the ML Monitor if you wish to use it in the same memory location every time.
5. Activate the routine with the given 'SYS' command. Since this has been made reversible, you may de-activate with the same call. (Don't save the program when the Unwedge is active).
6. Enter 'NEW' to clear the Basic portion. You are now ready to use 'APPEND/RENUM'.

TO RENUMBER A BASIC PROGRAM IN MEMORY:

1. Clear the screen, and move the cursor 3/4 of the way down the screen.
2. Enter the '<' key in the first column of a line, then hit the 'R' key.
3. If you hit 'RETURN' now, the program will be renumbered, starting at 100, in steps of 10. (The default conditions).
4. For a different starting point, simply enter that number next. Be careful that the final line number will not exceed 63999.
5. If you wish a different step size, enter a comma (','), then the new increment. Values up to 255 are acceptable.
6. After you have hit 'RETURN', the message 'RENUMBERING' will be printed. The screen memory is used for storing the line numbers, so it will now show a variety of characters.
7. When the cursor returns (5-20 seconds), the job is complete. All 'GOTO','THEN','GOSUB' and 'RUN' destinations will be changed. Any illegal or unreferenced line numbers will be numbered '65535' as a flag to the error.
8. The screen memory will handle only about 500 lines. This shouldn't be a real limitation on most programs.

TO APPEND ONE BASIC PROGRAM TO ANOTHER:

1. Place the program to be appended in Cassette #1 drive. The first line number of this program must be larger than the last one currently in memory. Use the Renumber function to prepare the program tape, if necesssary. (This is the benefit of combining these two utilities).
2. Enter the '<' key in the first column of a screen line, then the letter 'A'. (You may enter the whole word, 'APPEND', but the program only checks the first letter).
3. If you wish a specific file to be Appended from tape, enclose the file name in quotes, just as with a normal Load.
4. If there is Machine Language after the end of Basic (of program in memory), then the routine aborts with message: '?NOT ALL BASIC PROGRAM ERROR'.
5. If the program on tape is Machine Language, then this program simply Loads normally, but does not Append. The same message is printed, but without the 'ERROR'. Be careful here because the the Basic pointers will have been changed.
6. If the whole program will not fit in memory, the routine aborts. An 'OUT OF MEMORY ERROR' message will be printed.
7. The message 'APPENDING' will be printed, while the routine searches for the correct program.
8. The routine accounts for the differences between Original and Upgrade ROM. The Original ROM saved an extra byte after the end of Basic.

I hope that this proves useful for you. I am particulary indebted to Jim Butterfield, for his counsel and patience through many elementary questions.

David A. Hook
58 Steel Street
BARRIE, Ontario L4M 2E9

RELOCATING APPEND & RENUMBER


```
10 EA=PEEK(53)*256+PEEK(52):B=770:SA=EA-B:J%=SA/256:J=SA-J%*256
20 POKEEA-1,J%:POKE53,J%:POKE49,J%:POKEEA-2,J:POKE52,J:POKE48,J
30 J%=(SA+31)/256:J=SA+31-J%*256:POKEEA-3,J%:POKEEA-4,J
40 FORI=SATOEA-5:READA$:A=VAL(A$)
50 IFLEFT$(A$,1)="H"THENJ=I+VAL(MID$(A$,2)):A=J/256:GOTO70
60 IFLEFT$(A$,1)="L"THENJ=I+VAL(MID$(A$,2)):A=J+.05-256*INT(J/256)
70 POKEI,A:NEXT
100 PRINT"■ACTIVATE OR CANCEL ■APPEND/RENUM":PRINTTAB(20)"■--WITH: ■'SYS"SA"■'
110 PRINT"■SAVE WITH MLM:":PRINT"■.S"CHR$(34)"APPEND/RENUM"CHR$(34)",01";
120 X=SA/4096:GOSUB170:X=EA/4096:GOSUB170:PRINT:PRINT:END
170 PRINT",";:FORJ=1TO4:X%=X:X=(X-X%)*16:IFX%>9THENX%=X%+7
180 PRINTCHR$(X%+48);:NEXTJ:RETURN
10000 DATA173,L767,H766,133,52,173,L763,H762,133,53,162,3,181,120
10010 DATA72,189,L748,H747,149,120,104,157,L742,H741,202,208,241,76,121,197,96
10015 DATA201,60,208,8,72
10020 DATA 165,119,201,0,240,8,104,201,58,176,239,76,125,0,32,112
10030 DATA 0,201,65,240,7,201,82,208,237,76,L282,H281,162,1,134,212
10040 DATA 202,134,209,134,157,169,2,133,219,32,112,0,170,240,23,201
10050 DATA 34,208,246,166,119,232,134,218,32,112,0,170,240,8,201,34
10060 DATA 240,4,230,209,208,242,32,86,246,32,10,248,32,10,244,165
10070 DATA 240,4,230,209,208,242,32,86,246,32,10,248,32,10,244,165
10080 DATA 224,1,208,235,165,150,41,16,208,127,162,24,173,124,2,201
10090 DATA 4,240,7,162,0,32,L128,H127,240,108,32,L123,H122,56,165,42
10100 DATA 233,2,133,42,165,43,233,0,133,43,160,0,177,42,240,24
10110 DATA 32,L91,H90,32,110,242,169,13,32,210,255,169,63,32,210,255
10120 DATA 32,L91,H90,32,110,242,169,13,32,210,255,169,63,32,210,255
10130 DATAL63,H62,173,125,2,56,237,123,2,170,173,126,2,237,124,2
10140 DATA 168,165,42,56,233,2,141,123,2,165,43,233,0,141,124,2
10150 DATA 138,24,109,123,2,141,125,2,152,109,124,2,141,126,2,197
10160 DATA 53,144,3,76,85,195,32,185,243,76,221,243,32,L2,H1,230
10170 DATA 42,208,2,230,43,96,189,L11,H10,240,6,32,210,255,232,208
10180 DATA 245,96,13,78,79,84,32,65,76,76,32,66,65,83,73,67
10190 DATA 32,80,82,79,71,82,65,77,32,0,13,65,80,80,69,78
10200 DATA 68,73,78,71,32,0,13,82,69,78,85,77,66,69,82,73
10210 DATA 78,71,13,0,32,112,0,240,33,176,249,32,115,200,72,166
10220 DATA 17,164,18,134,62,132,63,104,240,24,32,112,0,240,19,176
10230 DATA 249,32,115,200,166,17,134,66,208,12,169,100,133,62,169,0
10240 DATA 133,63,169,10,133,66,162,36,32,L-115,H-116,169,254,133,33,169
10250 DATA 127,133,34,165,40,133,31,165,41,133,32,32,L292,H291,160,3
10260 DATA 177,31,145,33,185,92,0,145,31,136,192,1,208,242,177,31
10270 DATA 240,16,32,L301,H300,170,136,177,31,133,31,134,32,32,L267,H266
10280 DATA 208,220,169,255,200,145,33,200,145,33,165,40,133,119,165,41
10290 DATA 133,120,208,3,32,L277,H276,32,L274,H273,208,3,76,57,196,32
10300 DATAL266,H265,32,L263,H262,32,L260,H259,170,240,233,162,4,221,L262,H261
10310 DATA 240,5,202,208,248,240,238,165,119,72,165,120,72,32,112,0
10320 DATA 176,230,32,115,200,32,L51,H50,104,133,120,104,133,119,160,0
10330 DATA 162,0,189,1,1,240,15,72,32,112,0,144,3,32,L82,H81
10340 DATA 104,145,119,232,208,236,32,112,0,176,8,32,L102,H101,32,118
10350 DATA 0,144,248,201,44,240,192,208,175,32,L134,H133,169,0,133,33
10360 DATA 169,128,133,34,160,1,177,33,197,18,240,21,201,255,208,24
10370 DATA 133,95,133,94,165,94,133,96,162,144,56,32,85,219,76,233
10380 DATA 220,136,177,33,197,17,240,236,32,L96,H95,32,L116,H115,208,212
10390 DATA 32,L62,H61,160,0,177,31,200,145,31,32,L90,H89,208,8,230
10400 DATA 42,208,2,230,43,136,96,164,31,208,2,198,32,198,31,76
10410 DATAL-29,4-30,32,L28,H27,160,1,177,33,136,145,33,32,L56,H55,240
10420 DATA 5,32,L65,H64,208,239,164,42,208,2,198,43,198,42,96,165
10430 DATA 42,133,31,165,43,133,32,165,119,133,33,165,120,133,34,96
10440 DATA 165,62,133,94,165,63,133,95,96,165,34,24,101,66,133,94
10450 DATA 144,2,230,95,96,165,31,197,33,208,4,165,32,197,34,96
10460 DATA 32,L2,H1,230,33,208,2,230,34,96,160,0,230,119,208,2
10470 DATA 230,120,177,119,96,137,138,141,167,76
```

```
0010 ;*****************************************
0020 ;* APPEND AND RENUMBER BASIC PROGRAMS    *
0030 ;*                                       *
0040 ;* (C) DAVID A. HOOK, 58 STEEL STREET    *
0050 ;*       BARRIE, ONTARIO  L4M 2E9         *
0060 ;*                                       *
0070 ;*         CANADA (705) 726-8126         *
0080 ;*                                       *
0090 ;*****************************************
0095 ;
0100 ;BASIC VARIABLES
0110 ;
0120 TXTPTR      .DE $77        ;TEXTPOINTER
0130 BUF         .DE $0200      ;INPUT BUFFER
0140 VARTAB      .DE $2A        ;START VARIABLES
0150 MEMSIZ      .DE $34        ;TOP OF MEMORY
0160 TXTTAB      .DE $28        ;START OF BASIC
0170 ;
0200 ;BASIC ROUTINES
0210 ;
0220 CHRGET      .DE $70        ;GET NEXT CHAR
0230 CHRGOT      .DE $76        ;GET LAST CHAR
0240 FINI        .DE $C439      ;FIX BASIC LINKS
0250 CLR         .DE $C577      ;PERFORM 'CLR'
0260 LINGET      .DE $C873      ;GET LINE #
0270 ;
0300 ;OP SYSTEM VARIABLES
0310 ;
0320 FA          .DE $D4        ;DEVICE NUMBER
0330 VERCK       .DE $9D        ;LOAD OR VERIFY ?
0340 FNLEN       .DE $D1        ;FILENAME LENGTH
0350 FNADR       .DE $DA        ;FILENAME ADDRESS
0360 SATUS       .DE $96        ;STATUS BYTE
0370 TAPE1       .DE $027A      ;TAPE#1 BUFFER
0380 EAL         .DE $C9        ;END ADR LO
0390 EAH         .DE $CA        ;END ADR HI
0400 LINNUM      .DE $11        ;PLACE FOR LINE #
0410 SCREEN      .DE $8000      ;START SCREEN MEMORY
0420 ;
0450 ;OP SYSTEM ROUTINES
0460 ;
0470 ZZZ         .DE $F656      ;SET TAPE BUF PTR
0480 CSTE1       .DE $F812      ;TAPE MSG
0490 LD300       .DE $F40A      ;PRINT FILENAME
0500 FAF         .DE $F494      ;FIND SPECIFIC FILE
0510 FAH         .DE $F5A6      ;FIND ANY FILE
0520 OP160       .DE $F56E      ;PRINT 'NOT FOUND'
0530 MEMFUL      .DE $C355      ;MEMORY FULL ERROR
0540 ABRT        .DE $F26E      ;ABORT I/O ACTIVITY
0550 PRT         .DE $FFD2      ;PRINT CHAR
0560 ERRMSG      .DE $C377      ;ISSUE ERROR MSG
0570 CONTLD      .DE $F3DD      ;FINISH LOAD
0580 LD16        .DE $F3B9      ;LOAD FILE
0590 FLOATC      .DE $DB55      ;FLOAT A BINARY NUMBER
0600 FOUT        .DE $DCE9      ;MAKE ASCII
0640 ;
0650 ;PROGRAM VARIABLES
0660 ;
```

```
                        0670 BEGNUM     .DE 100         ;START LIN #
                        0680 INCNUM     .DE 10          ;STEP SIZE
                        0690 FACTO      .DE $5F         ;USED TO CONVERT NUMBERS
                        0700 INDEX1     .DE $1F         ;TEXT BUFFER
                        0710 INDEX2     .DE $21         ;SCREEN BUFFER
                        0720 STEP       .DE $42         ;STORE INCR
                        0730 BEGIN      .DE $3E         ;STORE START LINE #
                        0740 ;
                        0750 ;
                        0760 ;ADJUST MEMORY POINTERS AND EXCHANGE 'UNWEDGE' SEQUENCE
                        0770 ;
                        0780 ;
                        0800            .OS
                        0810            .BA $7000
7000- AD 00 73          0820            LDA MEMEND      ;GET MEMORY POINTERS
7003- 85 34             0830            STA *MEMSIZ
7005- AD 01 73          0840            LDA MEMEND+1
7008- 85 35             0850            STA *MEMSIZ+1
                        0870 ;
700A- A2 03             0890            LDX #3          ;PICK UP OR SWAP WEDGE INSTRCS
700C- B5 78             0900 LOOP       LDA *CHRGOT+2,X
700E- 48                0910            PHA
700F- BD FC 72          0920            LDA WG.REST-1,X
7012- 95 78             0930            STA *CHRGOT+2,X
7014- 68                0940            PLA
7015- 9D FC 72          0950            STA WG.REST-1,X
7018- CA                0960            DEX
7019- D0 F1             0970            BNE LOOP
701B- 4C 79 C5          0975            JMP CLR+2       ;RESET BASIC POINTERS
701E- 60                0980 STRTS      RTS  ;RETURN TO BASIC
                        0990 ;
                        0995 ;CHECK FOR 'UNWEDGE' COMMAND
                        1000 ;
                        1010 ;
701F- C9 3C             1020 WEDGE      CMP #'<         ;A WEDGE COMMAND ?
7021- D0 08             1030            BNE WG200       ;NO
7023- 48                1040            PHA  ;MAYBE
7024- A5 77             1050            LDA *TXTPTR     ;WAS '<' IN COLUMN 1
7026- C9 00             1060            CMP #L,BUF
7028- F0 08             1070            BEQ WCMD        ;YES-IT'S 'UNWEDGE'
                        1080 ;
702A- 68                1090 WG100      PLA  ;FINISH CHRGOT
702B- C9 3A             1100 WG200      CMP #':
702D- B0 EF             1110            BCS STRTS
702F- 4C 7D 00          1120            JMP CHRGOT+7
                        1130 ;
7032- 20 70 00          1140 WCMD       JSR CHRGET
7035- C9 41             1150            CMP #'A         ;AN APPEND?
7037- F0 07             1160            BEQ WG300       ;YES
7039- C9 52             1170            CMP #'R         ;A RENUMBER?
703B- D0 ED             1180            BNE WG100       ;NO
703D- 4C 58 71          1190            JMP RENUM       ;RENUMBER
                        1200 ;
                        1210 ;
                        1220 ;
7040- A2 01             1230 WG300      LDX #$01
7042- 86 D4             1240            STX *FA         ;SET CASSETTE#1
7044- CA                1250            DEX
7045- 86 D1             1260            STX *FNLEN      ;ZERO FILENAME LENGTH
```

```
7047- 86 9D      1270            STX *VERCK    ;SET TO LOAD
7049- A9 02      1280            LDA #$02
704B- 85 DB      1290            STA *FNADR+1          ;POINT TO BASIC INPUT BUFF
R
                 1300 ;
704D- 20 70 00   1320 WC100      JSR CHRGET    ;GET NEXT CHARACTER
7050- AA         1330            TAX
7051- F0 17      1340            BEQ WC210     ;IS THIS THE LAST ENTRY
7053- C9 22      1350            CMP #'"       ;START OF FILE NAME?
7055- D0 F6      1360            BNE WC100     ;NO--LOOP
7057- A6 77      1370            LDX *TXTPTR   ;START FILE NAME ONE PLUS
7059- E8         1380            INX
705A- 86 DA      1390            STX *FNADR    ;STARTS HERE
                 1400 ;
705C- 20 70 00   1420 WC200      JSR CHRGET    ;FIND END OF FILENAME
705F- AA         1430            TAX
7060- F0 08      1435            BEQ WC210     ;THE END?
7062- C9 22      1440            CMP #'"       ;END QUOTE?
7064- F0 04      1445            BEQ WC210
7066- E6 D1      1450            INC *FNLEN    ;NO--KEEP IT
7068- D0 F2      1460            BNE WC200     ;AND LOOP
                 1470 ;
706A- 20 56 F6   1490 WC210      JSR ZZZ       ;SET TAPE BUFFER
706D- 20 12 F8   1500            JSR CSTE1     ;GIVE TAPE MESSAGE
7070- 20 0A F4   1510            JSR LD300     ;PRT FILENAME
7073- A5 D1      1520 WC215      LDA *FNLEN    ;ANY FILE?
7075- F0 08      1530            BEQ WC250     ;YES
7077- 20 94 F4   1540            JSR FAF       ;FIND A SPECIFIC FILE
707A- D0 08      1550            BNE WC270     ;SKIP IF FOUND
707C- 4C 6E F5   1560 WC220      JMP OP160     ;NOT FOUND
707F- 20 A6 F5   1570 WC250      JSR FAH       ;READ ANY FILE
7082- F0 F8      1580            BEQ WC220     ;NOT FOUND?
7084- E0 01      1590 WC270      CPX #$01
7086- D0 EB      1600            BNE WC215     ;TRY AGAIN
7088- A5 96      1610            LDA *SATUS    ;CHECK FOR LOAD ERROR
708A- 29 10      1620            AND #%00010000
708C- D0 7F      1630            BNE WC300     ;IF ERROR
                 1670 ;
                 1675 ;
708E- A2 18      1680 WC280      LDX #$18      ;PREPARE TO GIVE 'APPENDING' MSG
                 1685 ;
7090- AD 7C 02   1690            LDA TAPE1+2   ;GET LOAD START ADDRESS
7093- C9 04      1700            CMP #$04      ;IS IT BASIC?
7095- F0 07      1710            BEQ AP100     ;YES--BRANCH
7097- A2 00      1720            LDX #$00      ;GET 'NOT ALL BASIC' MSG PTR
7099- 20 1A 71   1730            JSR WRMSG     ;PRT MSG
709C- F0 6C      1740            BEQ WC290     ;BRANCH ALWAYS
709E- 20 1A 71   1750 AP100      JSR WRMSG     ;PRT MSG
                 1755 ;
                 1760 ;
70A1- 38         1770            SEC  ;SUBTRACT 2 FROM START OF VARIABLES
70A2- A5 2A      1780            LDA *VARTAB
70A4- E9 02      1790            SBC #$02
70A6- 85 2A      1800            STA *VARTAB
70A8- A5 2B      1810            LDA *VARTAB+1
70AA- E9 00      1820            SBC #$00
70AC- 85 2B      1830            STA *VARTAB+1
70AE- A0 00      1840            LDY #$00
70B0- B1 2A      1850            LDA (VARTAB),Y
70B2- F0 18      1860            BEQ PT400     ;IF ZERO THEN NO EXTRA M.L. AFTER
```

```
                    1870 ;
                    1875 ;
70B4- 20 10 71      1890            JSR INVAR2    ;RESTORE VARIABLES POINTER
70B7- 20 6E F2      1900            JSR ABRT      ;ABORT THE APPEND/LOAD
70BA- A9 0D         1910            LDA #$0D
70BC- 20 D2 FF      1920            JSR PRT
70BF- A9 3F         1930            LDA #'?
70C1- 20 D2 FF      1940            JSR PRT
70C4- A2 01         1950            LDX #$01
70C6- 20 1A 71      1960            JSR WRMSG     ;ISSUE 'NOT ALL BASIC' MSG
70C9- 4C 77 C3      1970            JMP ERRMSG    ;TREAT AS AN ERROR CONDITION & EXIT
                    1980 ;
                    1990 ;
                    1995 ;CHECK IF CURRENT PROGRAM WAS SAVED ON OLD ROMS
                    2000 ;
                    2005 ;
70CC- 20 13 71      2010 PT400      JSR INVAR     ;BUMP POINTER
70CF- B1 2A         2020            LDA (VARTAB),Y
70D1- D0 03         2030            BNE WC285     ;BRANCH IF OLD
70D3- 20 13 71      2040            JSR INVAR     ;BUMP AGAIN IF NEW ROM
                    2045 ;
                    2050 ;
                    2055 ;CALC NEW ADDRESS FOR APPEND SOURCE
                    2060 ;
                    2070 ;
70D6- AD 7D 02      2080 WC285      LDA TAPE1+3   ;LO END
70D9- 38            2090            SEC
70DA- ED 7B 02      2100            SBC TAPE1+1   ;LO START
70DD- AA            2110            TAX   ;SAVE IN R(X)
70DE- AD 7E 02      2120            LDA TAPE1+4   ;HI END
70E1- ED 7C 02      2130            SBC TAPE1+2   ;HI START
70E4- A8            2140            TAY   ;SAVE IN R(Y)
                    2145 ;
70E5- A5 2A         2150            LDA #VARTAB   ;CALC APPEND BEGIN
70E7- 38            2160            SEC
70E8- E9 02         2170            SBC #$02      ;BACK UP OVER LAST TWO 'ZERO' BYTES
70EA- 8D 7B 02      2180            STA TAPE1+1   ;NEW LOAD START LO
70ED- A5 2B         2190            LDA #VARTAB+1
70EF- E9 00         2200            SBC #$00
70F1- 8D 7C 02      2210            STA TAPE1+2   ;NEW LOAD START HI
70F4- 8A            2220            TXA   ;GET DELTA LO
70F5- 18            2230            CLC
70F6- 6D 7B 02      2240            ADC TAPE1+1   ;CALC NEW LOAD END LO
70F9- 8D 7D 02      2250            STA TAPE1+3
70FC- 98            2260            TYA
70FD- 6D 7C 02      2270            ADC TAPE1+2   ;CALC NEW END HI
7100- 8D 7E 02      2280            STA TAPE1+4
7103- C5 35         2290            CMP #MEMSIZ+1         ;CHECK TO SEE IF MEM FULL
7105- 90 03         2300            BCC WC290     ;IF OK
                    2305 ;
7107- 4C 55 C3      2310            JMP MEMFUL    ;ERROR AND EXIT
                    2315 ;
                    2320 ;
                    2325 ;CONTINUE WITH LOAD OF FILE
                    2340 ;
                    2345 ;
710A- 20 B9 F3      2350 WC290      JSR LD16      ;OF SYSTEM LOAD
                    2360 ;
710D- 4C DD F3      2380 WC300      JMP CONTLD    ;CONTINUE LOAD
```

```
                        2450 ;
                        2460 ;
                        2500 ;
7110- 20 13 71          2510 INVAR2     JSR INVAR     ;BUMP TWICE
7113- E6 2A             2520 INVAR      INC *VARTAB   ;INCREMENT START OF VARIABLES POI
ER
7115- D0 02             2530            BNE IRTS
7117- E6 2B             2540            INC *VARTAB+1
7119- 60                2550 IRTS       RTS
                        2560 ;
                        2570 ;
                        2580 ;
711A- BD 26 71          2590 WRMSG      LDA MSG1,X    ;OUTPUT MESSAGE
711D- F0 06             2600            BEQ WRTS
711F- 20 D2 FF          2610            JSR PRT
7122- E8                2620            INX
7123- D0 F5             2630            BNE WRMSG
7125- 60                2640 WRTS       RTS
                        2650 ;
7126- 0D 4E 4F          2680 MSG1       .BY $0D 'NOT ALL BASIC PROGRAM ' $00
7129- 54 20 41
712C- 4C 4C 20
712F- 42 41 53
7132- 49 43 20
7135- 50 52 4F
7138- 47 52 41
713B- 4D 20 00
713E- 0D 41 50          2690            .BY $0D 'APPENDING ' $00
7141- 50 45 4E
7144- 44 49 4E
7147- 47 20 00
714A- 0D 52 45          2700            .BY $0D 'RENUMBERING' $0D $00
714D- 4E 55 4D
7150- 42 45 52
7153- 49 4E 47
7156- 0D 00
                        2705 ;
                        2710 ;
                        2720 ;RENUMBER BASIC PROGRAM
                        2725 ;
                        2800 ;
7158- 20 70 00          2810 RENUM      JSR CHRGET    ;GET CHAR
715B- F0 21             2820            BEQ START     ;DEFAULT IF NO NUMBER ENTERED
715D- B0 F9             2830            BCS RENUM     ;SPAN BLANKS
715F- 20 73 C8          2840            JSR LINGET    ;GET NUMBER AND CONVERT IN 'LINNU
7162- 48                2850            PHA  ;KEEP LAST CHAR
7163- A6 11             2860            LDX *LINNUM  ;LO START #
7165- A4 12             2870            LDY *LINNUM+1          ;HI START #
7167- 86 3E             2880            STX *BEGIN    ;SAVE START
7169- 84 3F             2890            STY *BEGIN+1           ;LINE NUMBER
716B- 68                2900            PLA  ;RESTORE LAST CHAR
716C- F0 18             2910            BEQ DFLT.INC           ;IF NO MORE CHARS
                        2915 ;
716E- 20 70 00          2920 INCR       JSR CHRGET    ;GET ANOTHER
7171- F0 13             2930            BEQ DFLT.INC           ;IF NO MORE
7173- B0 F9             2940            BCS INCR      ;WAIT FOR NUMBERS
7175- 20 73 C8          2950            JSR LINGET    ;GET NUMBER AND CONVERT IN 'LINNU
7178- A6 11             2960            LDX *LINNUM  ;GET LO INCREMENT
717A- 86 42             2970            STX *STEP     ;KEEP IT
717C- D0 0C             2980            BNE CDPNUM    ;BRANCH ALWAYS
```

```
                    2990 ;
                    3000 ;
                    3005 ;
717E- A9 64         3010 START      LDA #L,BEGNUM          ;DEFAULT STARTING LINE NUMB
ER=100
7180- 85 3E         3020            STA *BEGIN
7182- A9 00         3030            LDA #H,BEGNUM
7184- 85 3F         3040            STA *BEGIN+1
7186- A9 0A         3050 DFLT.INC   LDA #INCNUM  ;DEFAULT INCREMENT =10
7188- 85 42         3060            STA *STEP
                    3065 ;
                    3070 ;
                    3075 ;
718A- A2 24         3080 COPNUM     LDX #$24      ;PREPARE FOR 'RENUMBERING' MSG
718C- 20 1A 71      3090            JSR WRMSG
                    3100 ;
718F- A9 FE         3110            LDA #L,SCREEN-2       ;USE SCREEN AS BUFFER
7191- 85 21         3120            STA *INDEX2
7193- A9 7F         3130            LDA #H,SCREEN-2
7195- 85 22         3140            STA *INDEX2+1
7197- A5 28         3150            LDA *TXTTAB   ;BEGINNING OF TEXT
7199- 85 1F         3160            STA *INDEX1
719B- A5 29         3170            LDA *TXTTAB+1
719D- 85 20         3180            STA *INDEX1+1
719F- 20 C4 72      3190            JSR SETFAC    ;SET NEW # COUNTER
71A2- A0 03         3200 COP10      LDY #3
71A4- B1 1F         3210 COP20      LDA (INDEX1),Y        ;GET LINE NUMBER
71A6- 91 21         3220            STA (INDEX2),Y        ;SAVE IT
71A8- B9 5C 00      3230            LDA FACTO-3,Y         ;GET NEW NUMBER
71AB- 91 1F         3235            STA (INDEX1),Y        ;REPLACE OLD
71AD- 88            3240            DEY
71AE- C0 01         3250            CPY #1        ;POINT AT LINK?
71B0- D0 F2         3260            BNE COP20     ;NO--LOOP
71B2- B1 1F         3270            LDA (INDEX1),Y        ;LAST LINE?
71B4- F0 10         3280            BEQ COP80     ;YES--DONE
71B6- 20 E4 72      3290            JSR BUPX2     ;NO--BUMP POINTER 2
71B9- AA            3300            TAX  ;SAVE HI LINK
71BA- 88            3310            DEY
71BB- B1 1F         3320            LDA (INDEX1),Y        ;GET LO
71BD- 85 1F         3330            STA *INDEX1   ;MAKE NEW POINTER
71BF- 86 20         3340            STX *INDEX1+1
71C1- 20 CD 72      3350            JSR ADDSTP    ;CALC NEW #
71C4- D0 DC         3360            BNE COP10     ;BRANCH ALWAYS
                    3370 ;
71C6- A9 FF         3380 COP80      LDA #$FF      ;MAKE LAST BIGGEST
71C8- C8            3390            INY
71C9- 91 21         3400            STA (INDEX2),Y        ;FINISH IN BUFFER
71CB- C8            3410            INY
71CC- 91 21         3420            STA (INDEX2),Y
71CE- A5 28         3430 RENN       LDA *TXTTAB   ;START CHRGET AT SOURCE
71D0- 85 77         3440            STA *TXTPTR
71D2- A5 29         3450            LDA *TXTTAB+1
71D4- 85 78         3460            STA *TXTPTR+1
71D6- D0 03         3470            BNE RN15      ;GET HI LINK
                    3480 ;
71D8- 20 EE 72      3490 RN10       JSR GRAB      ;SKIP LO LINK
71DB- 20 EE 72      3500 RN15       JSR GRAB      ;GET HI LINK
71DE- D0 03         3510            BNE RN20      ;SKIP IF <>0
                    3515 ;
71E0- 4C 39 C4      3520            JMP FINI      ;DONE--FIX LINKS
```

```
                        3535 ;
71E3- 20 EE 72   3540 RN20    JSR GRAB      ;SKIP LO LINE #
71E6- 20 EE 72   3550         JSR GRAB      ;SKIP HI LINE #
71E9- 20 EE 72   3560 RN30    JSR GRAB      ;GET BASIC SOURCE
71EC- AA         3570 RN35    TAX  ;END OF LINE?
71ED- F0 E9      3580         BEQ RN10      ;YES--LOOP
                        3590 ;
71EF- A2 04      3600 RN40    LDX #4        ;LOOK FOR 4 TOKENS
71F1- DD F8 72   3610 RN45    CMP TOKEN-1,X
71F4- F0 05      3620         BEQ RN50      ;IF FOUND
71F6- CA         3630         DEX
71F7- D0 F8      3640         BNE RN45      ;KEEP LOOKING
71F9- F0 EE      3650         BEQ RN30      ;NOT FOUND
                        3655 ;
71FB- A5 77      3660 RN50    LDA *TXTPTR   ;SAVE POINTER
71FD- 48         3670         PHA
71FE- A5 78      3680         LDA *TXTPTR+1
7200- 48         3690         PHA
7201- 20 70 00   3700         JSR CHRGET    ;CHECK TO SEE IF A NUMBER IS NEXT
7204- B0 E6      3710         BCS RN35
7206- 20 73 C8   3720         JSR LINGET    ;YES--MAKE BINARY
7209- 20 3D 72   3730         JSR FINNUM    ;FIND OLD, MAKE NEW
720C- 68         3740         PLA  ;MOVE PTR TO NUMBER START
720D- 85 78      3750         STA *TXTPTR+1
720F- 68         3760         PLA
7210- 85 77      3770         STA *TXTPTR
7212- A0 00      3780         LDY #0        ;SET INDEX
7214- A2 00      3790         LDX #0
7216- BD 01 01   3800 RN60    LDA $0101,X   ;GET ASCII NEW NUMBER
7219- F0 0F      3810         BEQ RN70      ;DONE IF ZERO
721B- 48         3820         PHA
721C- 20 70 00   3830         JSR CHRGET    ;IS OLD CHARACTER ASCII?
721F- 90 03      3840         BCC RN65      ;YES--SKIP TO REPLACE
7221- 20 74 72   3850         JSR MOVUP     ;NO MOVE UP TEXT
7224- 68         3860 RN65    PLA  ;GET SAVED NEW #
7225- 91 77      3870         STA (TXTPTR),Y        ;REPLACE OLD #
7227- E8         3880         INX
7228- D0 EC      3890         BNE RN60      ;BRANCH ALWAYS
                        3895 ;
722A- 20 70 00   3900 RN70    JSR CHRGET
722D- B0 08      3910 RN75    BCS RN80      ;NO--DONE
722F- 20 96 72   3920 RN78    JSR MOVDWN    ;YES--MOVE DOWN 1
7232- 20 76 00   3930         JSR CHRGOT    ;TEST NEW CHAR
7235- 90 F8      3940         BCC RN78
7237- C9 2C      3950 RN80    CMP #',       ; IS IT 'ON'?
7239- F0 C0      3960         BEQ RN50      ;YES--DO REST
723B- D0 AF      3970         BNE RN35      ;NO--LOOP
                        3975 ;
                        3980 ;
                        3985 ;FIND OLD NUMBER IN BUFFER, GENERATE NEW LINE #
                        3990 ;PUT IT IN ASCII, END WITH '00'
                        3995 ;
                        4000 ;
723D- 20 C4 72   4010 FINNUM  JSR SETFAC
7240- A9 00      4020         LDA #L,SCREEN
7242- 85 21      4030         STA *INDEX2
7244- A9 80      4040         LDA #H,SCREEN
7246- 85 22      4050         STA *INDEX2+1
7248- A0 01      4060 FN10    LDY #1
724A- B1 21      4070         LDA (INDEX2),Y
```

```
724C- C5 12      4080              CMP *LINNUM+1
724E- F0 15      4090              BEQ FN50
7250- C9 FF      4100              CMP #$FF
7252- D0 18      4110              BNE FN60
7254- 85 5F      4120              STA *FACTO
7256- 85 5E      4130              STA *FACTO-1
7258- A5 5E      4140 FN20         LDA *FACTO-1
725A- 85 60      4150              STA *FACTO+1
725C- A2 90      4160              LDX #$90
725E- 38         4170              SEC
725F- 20 55 DB   4180              JSR FLOATC    ;FLOAT #
7262- 4C E9 DC   4190              JMP FOUT      ;CONVERT TO ASCII
                 4200 ;
7265- 88         4210 FN50         DEY
7266- B1 21      4220              LDA (INDEX2),Y
7268- C5 11      4230              CMP *LINNUM
726A- F0 EC      4240              BEQ FN20
726C- 20 CD 72   4250 FN60         JSR ADDSTP
726F- 20 E4 72   4260              JSR BUPX2
7272- D0 D4      4270              BNE FN10
                 4280 ;
                 4285 ;
                 4290 ;MOVE TEXT UP ONE BYTE
                 4300 ;
                 4305 ;
7274- 20 B3 72   4310 MOVUP        JSR SETPTR
7277- A0 00      4320 MU10         LDY #0
7279- B1 1F      4330              LDA (INDEX1),Y
727B- C8         4340              INY
727C- 91 1F      4350              STA (INDEX1),Y
727E- 20 D9 72   4360              JSR CMPX
7281- D0 08      4370              BNE MU40
7283- E6 2A      4380              INC *VARTAB
7285- D0 02      4390              BNE MU20
7287- E6 2B      4400              INC *VARTAB+1
7289- 88         4410 MU20         DEY
728A- 60         4420              RTS
                 4430 ;
728B- A4 1F      4440 MU40         LDY *INDEX1
728D- D0 02      4450              BNE MU60
728F- C6 20      4460              DEC *INDEX1+1
7291- C6 1F      4470 MU60         DEC *INDEX1
7293- 4C 77 72   4480              JMP MU10
                 4485 ;
                 4490 ;
                 4495 ;MOVE TEXT DOWN ONE BYTE
                 4500 ;
                 4510 ;
7296- 20 B3 72   4520 MOVDWN       JSR SETPTR
7299- A0 01      4530 MD10         LDY #1
729B- B1 21      4540              LDA (INDEX2),Y
729D- 88         4550              DEY
729E- 91 21      4560              STA (INDEX2),Y
72A0- 20 D9 72   4570              JSR CMPX
72A3- F0 05      4580              BEQ MD30
72A5- 20 E7 72   4590 MD20         JSR BUPX1
72A8- D0 EF      4600              BNE MD10
72AA- A4 2A      4610 MD30         LDY *VARTAB
72AC- D0 02      4620              BNE MD35
72AE- C6 2B      4630              DEC *VARTAB+1
```

```
72B0- C6 2A      4640 MD35       DEC *VARTAB
72B2- 60         4650 MD40       RTS
                 4660 ;
                 4680 ;
                 4682 ;SET POINTERS FOR TEXT MOVES
                 4685 ;
                 4687 ;
72B3- A5 2A      4690 SETPTR     LDA *VARTAB
72B5- 85 1F      4700            STA *INDEX1
72B7- A5 2B      4710            LDA *VARTAB+1
72B9- 85 20      4720            STA *INDEX1+1
72BB- A5 77      4730            LDA *TXTPTR
72BD- 85 21      4740            STA *INDEX2
72BF- A5 78      4750            LDA *TXTPTR+1
72C1- 85 22      4760            STA *INDEX2+1
72C3- 60         4770            RTS
                 4775 ;
                 4780 ;
                 4785 ;SET OP FACTO & FACTO-1 TO GENERATE NEW LINE NUMBERS
                 4790 ;
                 4800 ;
72C4- A5 3E      4810 SETFAC     LDA *BEGIN
72C6- 85 5E      4820            STA *FACTO-1
72C8- A5 3F      4830            LDA *BEGIN+1
72CA- 85 5F      4840            STA *FACTO
72CC- 60         4850            RTS
                 4855 ;
                 4860 ;
                 4865 ;ADDS STEP TO FACTO-1 & FACTO
                 4870 ;
                 4875 ;
72CD- A5 5E      4880 ADDSTP     LDA *FACTO-1
72CF- 18         4890            CLC
72D0- 65 42      4900            ADC *STEP
72D2- 85 5E      4910            STA *FACTO-1
72D4- 90 02      4920            BCC ARTS
72D6- E6 5F      4930            INC *FACTO
72D8- 60         4940 ARTS       RTS
                 4950 ;
                 4955 ;
                 4960 ;SET ZERO FLAG IF INDEX1=INDEX2
                 4965 ;
                 4970 ;
72D9- A5 1F      4980 CMPX       LDA *INDEX1
72DB- C5 21      4990            CMP *INDEX2
72DD- D0 04      5000            BNE CRTS
72DF- A5 20      5010            LDA *INDEX1+1
72E1- C5 22      5020            CMP *INDEX2+1
72E3- 60         5030 CRTS       RTS
                 5040 ;
                 5045 ;
                 5055 ;INCREMENTS INDEX2 (1 OR 2)
                 5060 ;
                 5065 ;
72E4- 20 E7 72   5070 BUPX2      JSR BUPX1
72E7- E6 21      5080 BUPX1      INC *INDEX2
72E9- D0 02      5090            BNE BRTS
72EB- E6 22      5100            INC *INDEX2+1
72ED- 60         5110 BRTS       RTS
                 5115 ;
```

```
                    5120 ;
                    5135 ;GET A CHARACTER & SET ZERO FLAG
                    5140 ;
                    5145 ;
72EE- A0 00         5150 GRAB       LDY #0
72F0- E6 77         5160            INC *TXTPTR
72F2- D0 02         5170            BNE GR10
72F4- E6 78         5180            INC *TXTPTR+1
72F6- B1 77         5190 GR10       LDA (TXTPTR),Y
72F8- 60            5200            RTS
                    5205 ;
                    5210 ;
                    5215 ;TOKENS FOR 'GOTO' 'RUN' 'GOSUB' 'THEN'
                    5220 ;
                    5225 ;
72F9- 89 8A 8D      5240 TOKEN      .BY $89 $8A $8D $A7
72FC- A7
                    5242 ;
                    5245 ;
                    5250 ;JUMP ADDRESS FOR UNWEDGE
                    5255 ;
72FD- 4C 00 00      5260 WG.REST    .BY $4C $00 $00
                    5262 ;
                    5265 ;
                    5270 ;STORAGE FOR END OF MEMORY
                    5275 ;
7300-               5280 MEMEND     .DS 2
                    5290 ;
                    9995            .EN
```

LABEL FILE: [/ = EXTERNAL]

```
/TXTPTR=0077        /BUF=0200           /VARTAB=002A
/MEMSIZ=0034        /TXTTAB=0028        /CHRGET=0070
/CHRGOT=0076        /FINI=C439          /CLR=C577
/LINGET=C873        /FA=00D4            /VERCK=009D
/FNLEN=00D1         /FNADR=00DA         /SATUS=0096
/TAPE1=027A         /EAL=00C9           /EAH=00CA
/LINNUM=0011        /SCREEN=8000        /ZZZ=F656
/CSTE1=F812         /LD300=F40A         /FAF=F494
/FAH=F5A6           /OP160=F56E         /MEMFUL=C355
/ABRT=F26E          /PRT=FFD2           /ERRMSG=C377
/CONTLD=F3DD        /LD16=F3B9          /FLOATC=DB55
/FOUT=DCE9          /BEGNUM=0064        /INCNUM=000A
/FACTO=005F         /INDEX1=001F        /INDEX2=0021
/STEP=0042          /BEGIN=003E         LOOP=700F
STRTS=701E          WEDGE=701F          WG100=702A
WG200=702B          WCMD=7032           WG300=7040
WC100=704D          WC200=705C          WC210=706A
WC215=7073          WC220=707C          WC250=707F
WC270=7084          WC280=708E          AP100=709E
PT400=70CC          WC285=70D6          WC290=710A
WC300=710D          INVAR2=7110         INVAR=7113
IRTS=7119           WRMSG=711A          WRTS=7125
MSG1=7126           RENUM=7158          INCR=716E
START=717E          DFLT.INC=7186       COPNUM=718A
COP10=71A2          COP20=71A4          COP80=71C6
RENN=71CE           RN10=71D8           RN15=71DB
RN20=71E3           RN30=71E9           RN35=71EC
```

```
RN40=71EF       RN45=71F1       RN50=71F3
RN60=7216       RN65=7224       RN70=722A
RN75=722D       RN78=722F       RN80=7237
FINNUM=723D     FN10=7248       FN20=7258
FN50=7265       FN60=726C       MOVUP=7274
MU10=7277       MU20=7289       MU40=728B
MU60=7291       MOVDWN=7296     MD10=7299
MD20=72A5       MD30=72AA       MD35=72B0
MD40=72B2       SETPTR=72B3     SETFAC=72C4
ADDSTP=72CD     ARTS=72D8       CMPX=72D9
CRTS=72E3       BUPX2=72E4      BUPX1=72E7
BRTS=72ED       GRAB=72EE       GR10=72F6
TOKEN=72F9      WG.REST=72FD    MEMEND=7300

//0000,7302,7302
```

RESTORE DATA Line Program

Paul Barnes
Deseronto, Ontario

Executing the RESTORE command causes the next READ to occur at the very first DATA element in your program. This subroutine can be used to RESTORE the DATA line pointer at a line other than the first.

It doesn't matter if you don't give the number of the line that has the "DATA" keyword in it that you want to start at, as long as it is past previous DATA statements so that the next data to be read will be the one desired.

```
  4 REM ********************************
  5 REM ***  RESTORE DATA LINE PGM  ***
  6 REM ***       BY PAUL BARNES     ***
  7 REM ***    DESERONTO, ONTARIO    ***
  8 REM ********************************
 10 DATA 166, 142, 134,   8, 166, 143
 10 DATA 166,  60, 134,  17, 166,  61
 15 DATA 134,   9,  32,  34, 197, 144
 15 DATA 134,  18,  32,  44, 197, 144
 20 DATA  11, 166, 174, 142, 132,   3
 20 DATA  11, 166,  92, 142, 132,   3
 25 DATA 166, 175, 142, 133,   3   96
 25 DATA 166,  93, 142, 133,   3   96
 30 DATA 162,   0, 142, 132,   3, 162
 35 DATA   0,  42, 133,   3,  96
 40 FOR F = 826 TO 860 : READ S : POKE
F, S : NEXT
 50 DATA "GOOD-BYE!"
 60 DATA "ANYBODY HOME?"
 70 J = 26545 * 10 : FOR D = 1 TO 100 :
DATA "MAYBE!"
 80 NEXT : DATA "HI!"
100 DATA "GO HOME!"
110 DATA "GO DIRECTLY TO JAIL!"
120 DATA "DO NOT PASS GO!"
130 DATA "DO NOT COLLECT $100!!!"
200 GOSUB 1000
210 FOR T = 1 TO 3 : READ A$ : PRINT A$
230 NEXT : PRINT : GOTO 200
998 REM *** SUBROUTINE TO RESTORE DATA
999 REM ***  AT A CERTAIN LINE NUMBER
1000 INPUT "RESTORE TO LINE"; A
1010 H = INT (A/256) : L = A - H * 256
1020 REM POKE CURRENT DATA LINE POINTER
1030 POKE 142, L : POKE 143, H
1030 POKE  60, L : POKE  61, H
1040 SYS 826
1050 L = PEEK(900) : H = PEEK(901)
1060 IF L=0 AND H=0 THEN PRINT "LINE NOT
FOUND" : GOTO 1000
1070 REM POKE MEMORY ADDRESS OF DATA LINE
1080 A = H * 256 + L - 1 : H = INT(A/256)
: L = A - H * 256
1090 POKE 144, L : POKE 145, H
1090 POKE  62, L : POKE  63, H
1100 RETURN
```

Editors Note: Paul has submitted the above program for the Original ROM set. The duplicate underlined statements are for BASIC 2.0 ROM.

A First Look at the Visible Music Monitor

W.T. Carbutt,
Mississauga, Ont.

It all started in September 1977! Hal Chamberlin produced the classic work on 6502 generated music in BYTE. This article was recently updated in the April 1980 issue of BYTE to the current state of the art for PET with the programming help from Dr. F. Covitz and Cliff Ashcraft.

Unfortunately until very recently the 'Chamberlin' style music has suffered from a serious problem , human interfacing, a problem that incidentally plagues a number of micro-computer programs. Readers of The Transactor in the Metro Toronto area are no doubt aware of Mike Bonnycastle's significant work to correct this problem, initially with an 'old ROM' version about a year ago and more recently with an 'upgrade ROM' version. Mike's pioneering efforts should not be overlooked as it still offers an effective human interfaced implementation, albeit the music graphics are non-standard and not available on playback and the editing is limited. Mike's work nevertheless issued a challenge to serious PET music users to build a better music mousetrap.

Well someone has taken that challenge seriously and what they have produced is a quantum leap forward in this temporal micro-computer world of daily future shock!

The product is called THE VISIBLE MUSIC MONITOR. It is programmmed by Dr. F. Levinson and is marketed by Gene Beal's AB COMPUTERS. (Readers of PET USER's NOTES will remember Gene as the editor.)

The following is a brief outline of the major features of this superb package (we'll take a closer user look at it in a future Transactor):

1. Written almost entirely in machine language to operate on any PET, 'old' or 'upgrade' ROM , 8k or greater.

2. Requires a simple user port 8-bit DAC, eg. as manufactured by AB COMPUTERS, Micro Technology Unlimited or Jim Butterfield's Poor Man's DAC as published in an earlier Transactor.

3. Full seven octave range and four voice polyphony.

4. Comphrensive screen graphics with full edit, cursor left/right , delete and insert.

5. Music graphics available on playback.

6. Auto baring.

7. Initial and subsequent key and time signature setting.

8. Simple two or three keystroke note entry including the provision to program your own keyboard entry syntax.

9. Adjustable dynamics (volume).

10. Preset or Fourier derived voices.

11. User programmable sequencer that permits scoring music in phrases and playing those phrases in user specified order during playback.

12. Adjustable tempo during playback.

13. Individually selectable vioce duration including dotted and triplet whole to 64th notes.

14. Full transposition.

15. Competively priced at $29.90 US.

What does the VISIBLE MUSIC MONITOR lack. Well you don't have tie notes. Accidentals are not carried throughout a bar requiring repeated user programming. The first and seventh octave are shown as the 2nd and 6th octave albeit with appropriate identifiers. The graphics are not entirely standard.

Finally chord entry is not the familiar 'one over the other' style but rather step horizontally across the screen. As each voice is tagged with its appropriate identification number along the bottom of the staff this method is actually helpful in entering one's music. On playback however the non-standard notation takes some getting used to.

Will AB Computer (or others) stand still? I doubt it. In the interim if you our looking for a definitive music monitor do yourself a favor and audition this package. For availability contact the following:

AB COMPUTERS,
115 E. Stump Road,
Montgomeryville, Pa.,
18936,
(215) 699-8386

REFERENCES

1. Chamberlin, Hal, "A Sampling of Techniques for Computer Performance of Music," September 1977 BYTE, pages 62 thru 83.(Reprinted in the BYTE Book of Computer Music, edited by Christopher P. Morgan, Byte Books, 1979, pages 47 thru 63.)

2. Chamberlin , Hal, "Advanced Real-Time Music Synthesis Techniques", pages 70-94, 180-196.

3. Butterfield, Jim, "Poor Man's D/A Converter", The Best of The Transactor, Page 58.

This machine language program will convert strings to the correct upper/lower case condition for printing on CBM 2022/23 printers with an original ROM PET. It is relocatable so will operate anywhere in memory. The routine given here puts it in the second cassette buffer, but changing the location given in line 10100 will place it wherever you wish.

There are several things which must be done in order for the routine to operate correctly. These are best demonstrated by the following program.

```
0  ML$="" : GOSUB 10000
10 POKE 59468, 14
20 ML$="azl23AZ"
30 PRINT ML$ : OPEN 4,4 : PRINT#4,ML$
40 SYS 826
50 PRINT#4,ML$ : CLOSE 4
60 PRINT ML$
70 LIST
10000 DATA 160,   2, 177, 124, 141, 251
10010 DATA   0, 200, 177, 124, 141, 252
10020 DATA   0, 200, 177, 124, 141, 253
10030 DATA   0, 172, 251,   0, 136, 177
10040 DATA 252, 201, 219, 176,  22, 201
10050 DATA 193, 144,   5,  56, 233, 128
10060 DATA 208,  11, 201,  65, 144,   9
10070 DATA 201,  91, 176,   5,  24, 105
10080 DATA 128, 145, 252, 192,   0, 208
10090 DATA 223,  96
10100 FOR A = 826 TO 881 : READ B
10110 POKE A, B : NEXT : RETURN
```

Note that line 20 is altered once the program is RUN. This is done by the SYS command in line 40.

Now alter line 20 to:

```
20 ML$ = ML$ + "azl23AZ"
```

and reRUN from line 0. This time line 20 has not been changed in the listing. Whenever a string is formed by concatenation, the new string is stored in a location different from the original strings i.e. up in high RAM. It is this new location that has been altered. The major advantage in working on a string stored away from the program listing is that you don't have to worry if the string has been previously altered.

Now change line 0 to:

```
0  A = 0 : GOSUB 10000
```

and reRUN from line 0. Two points to note are:

1. Make sure that the variable string to be printed is #1 in the variable table, and
2. form the string to be printed by concatenation.

```
                    MOVE VARIABLE POINTERS TO ZERO PAGE

LDY 2       Set Y register offset.
LDA 124,Y   Load A with byte from variable table pointed to by
            124/125 + Y
STA 251,0   and move to location 251.  This byte is the
            character count.
INY         Increment offset.
LDA 124,Y
STA 252,0
INY         Shift start address of string to zero page.
LDA 124,Y
STA 253,0

                          ADJUST STRING

LDY 251,0   Load Y with string character count from location 251.
DEY         Decrement Y offset.  Y points to character to be
            altered next.

                      TEST FOR LOWER CASE

LDA 252,Y   Load A with string byte pointed to by 252/253
            and offset by Y
CMP 219     and compare to lower case 'z' and
BCS 22      if greater than, skip to COMPARE Y.
CMP 193     Compare to lower case 'a' and
BCC 5       if less than skip to TEST FOR UPPER CASE.

                      ADJUST LOWER CASE

SEC         Set carry flag
SBC 128     Subtract 128 from the string byte in A and
BNE 11      always skip to STORE MODIFIED CHARACTER.

                      TEST FOR UPPER CASE

CMP 91      Compare to upper case 'Z' and
BCS 9       skip to COMPARE Y if greater than.
CMP 65      Compare to upper case 'A' and
BCC 5       skip to COMPARE Y if less than.

                      ADJUST UPPER CASE

CLC         Clear carry flag.
ADC 128     Add 128 to string byte.

                    STORE MODIFIED CHARACTER

STA 128,Y   Store byte at location pointed to by
            252/253 and offset by Y.

                    COMPARE Y FOR STRING END

CPY 0       Compare Y to '0' and
BNE 223     skip to DEY in ADJUST STRING if string
            not finished.
RTS         Otherwise, return to BASIC.
```

Without fanfare General Instruments introduced an LSI chip in the early Spring of 1979 called the Programmable Sound Generator (PSG). The chip was primarily intended for sound effects generation under micro-computer control. This facility clearly seperated it from the Texas Instruments Complex Sound Generator. The potential of computer control was (and is) enormous. Make that memory mapped register addressing , which the PSG is, and the benefits become nothing short of spectacular.

With equal understatement Steve Ciarcia published a comparison of the GI and Ti products last July (Byte, July 1979 , "Ciarcia's Circuit Cellar: Sound Off"); well as best a comparison can be between apples and oranges.

Imagine if you can a single 24 or 40 pin device (called the AY-3-8912 or AY-3-8910 respectively); operating off a single 5 volt power supply; requiring a minimum of external components (a 6520 as far as the PET is concerned)--the pull up resistors are for the most part included in the chip ; containing three DAC's, three envelope generators, three VCA's, three mixers; an noise generator and one or two 8 bit parallel user ports depending on which pin-out you need. As we mentioned the whole works is controlled by a memory mapped register addressing scheme requiring minimum processor overhead.

Performance specifications you ask? How about a range of eight octaves, a possible 45+db S/N, an optimum clock frequency of 1 mega-hertz, etc. The price? An incredibly low $15 or $20 depending on pin-out.

One enterprising firm from Chicago, ACKERMAN DIGITAL SYSTEMS INC (a company you should be hearing a great deal more about in the future) were not caught sleeping when the PSG arrived. ACKERMAN DIGITAL quickly produced an S-100 board called the NOISEMAKER, incorporating two of the 40 pin version! Imagine, six channel mono or full three channel stereo.

Any PET owner with an S-100 adapter can acquire an NOISEMAKER PCB from ACKERMAN DIGITAL for a mere 39.95 U.S.. For less than $100 (assuming a well stocked parts bin) and a modest programming skill one can have a super PET music synthesizer.

That's not all though!

If you are not up to the software challenge ACKERMAN DIGITAL has developed an 8080 music compiler for the NOISEMAKER. (We will see if a 6502 version can be derived.) ACKERMAN DIGITAL is also manufacturing a NOISEMAKER for

another well known 6502 system (was that Apple or Atari?). They are also developing a software package!

PET plans? You bet! The Transactor will keep you up to date with the latest progress on this milestone product.

For more information please contact,
ACKERMAN DIGITAL SYSTEMS INC.,
110 N. YORK Rd.,
Suite 208,
Elmhurst, Il. 60126

The notes below refer to articles in previous Transactors that contained errors or may require some clarity in explanation or application. Although everything published is tested, the possibility of an unforeseen condition or situation always exists. Most of the items printed in Back - Up were recognized by our readers...your feedback is appreciated.

Block Get - Transactor #7

By now I'm sure any programmer using direct disk access is using Block Get (Many thanks to Bill MacLean for this excellent routine). However, Block Get can also be used for sequential file access to recover strings of length greater than 80. Even 255 character strings can be retrieved with Block Get provided they are followed by a carriage return (CHR$(13)). Essentially, Block Get is the same as INPUT# but with a 255 character input buffer....which brings us to point 2.

This 255 byte buffer is set up in the very top page of RAM; $7F00 to $7FFF on 32Ks or $3F00 to $3FFF for 16Ks. This space must be sealed off before INPUTing or INPUT#ing strings, defining strings as the result of a concatenation, LOADing DOS Support or anything else that resides in this memory space. Otherwise when Block Get is called, the data will be transfered from the disk into the buffer and clobber your DOS Support, strings or whatever happens to be there.

```
POKE 53 , PEEK (53) - 1 : CLR
```

Location 53 ($35) is the high order byte of the Top Of BASIC Pointer. Decrementing 53 by 1 brings the pointer down by 256 thus "sealing off" the top page of memory. PET will then ignore this memory as though it's not even there (try ?FRE(0)). You may want to use absolute values rather than PEEK (53) - 1 since each time this is executed, the pointer will decrement another 256 bytes.

```
32K :   POKE 53 , 127 : CLR

16K :   POKE 53 , 63  : CLR
```

The CLR command equates some other pointers to the new value of the Top of BASIC Pointer i.e. the Bottom of Strings Pointer and the Utility String Pointer. These could also be POKEd, but CLR does the job quite nicely.

If DOS Support is to be used with Block Get, this statement should be executed prior to RUNning DOS. However Block Get contains one gotcha that will leave DOS open for certain destruction.

When DOS Support is LOADed and RUN, it sets itself up just below the Top of BASIC Pointer (TBP). After executing the above command, the TBP will now be 256 bytes lower.... but that's ok since DOS can live anywhere. Once set up

though, DOS lowers this pointer again to protect itself. But each time Block Get is called, the pointer is moved back to 256 bytes lower than the TBP at power up. Now DOS is sitting in memory that is available to BASIC. Re-RUN your program and whammo!...DOS Support gets clobbered by strings. Hit ">" and PET JSRs to where DOS used to be which is now ASCII characters....crash!

Fortunately this can be avoided. The first 8 bytes of Block Get sets the TBP every time it's called:

```
033A  LDA #$00
033C  STA  $34
033E  LDA #$7F   (3F for 16K)
0340  STA  $35
```

Therefore when using DOS Support and Block Get, SYS past these bytes with:

```
SYS 834 , LF# , A$

Instead of: SYS 826 , LF# , A$
```

Supermon 1.0 - Transactor #8

Those who have entered the code for Supermon 1.0 may have noticed an error when using the Checksum program i.e. BLOCK 42. The error lies not in Supermon but in the Checksum program. Two changes to make:

1. the last number in the Checksum program from 4501 to 5392.
2. the end addresses of the monitor Save commands on page 20 from 0D45 to 0D46.

Re-DIMensioning Arrays - Transactor 8

One extra note on re-dimensioning some arrays but not all (pg 2). Once the End of Arrays Pointer (EAP) is stored in PL% and PH% (line 100), no new variables should be defined. This includes all variable types; floating point, integer, string and array.

For a program, RAM is used by PET in the following order:

```
BASIC Text
Variables table
Arrays
    .
    .
    .
Dynamic Strings
```

Notice that Arrays immediately follow the Variables table. If a new variable (one that hasn't yet been executed) is defined after storing the EAP, the Arrays must all be moved up to make room for the new variable in the Variables table.

The EAP will now be slightly higher than what was stored previously. Therefore restoring the EAP to PL% and PH% will chop off part of the last permanent array.

To avoid any problems involving this, simply set all variables to zero (including PL% and PH%) before storing the EAP. For example, page 2 of Transactor #8 should include:

```
90  PL%=0 : PH%=0 : I%=0 : J%=0 : K%=0
```

Also, line 2100 should be:

```
2100  REM  RE-DIM ARRAYS X , Y% AND Z$
```

Cursor Positioning - Transactor 8

The following is an example for remembering and restoring the position of the cursor which should have been included in the last issue. First code the two subroutines as shown on page 3, Transactor 8.

```
100 PRINT "[CS]";
110 PRINT "[5DN]";
120 PRINT "THE POSITION OF THE CURSOR
WILL BE STORED HERE :"; : GOSUB 30050
130 PRINT "[HM]";
140 PRINT "THEN THE STRING '*****' WILL
BE PRINTED FOLLOWING THE COLON";
150 GOSUB 30150
160 PRINT "*****"; : END
```