Gee, I Wish I'd Been Nicer to Aunt Tillie... When Aunt Tillie died, did you feel guilty for neglecting her, for all the birthday cards you'd missed, the Christmas gifts unsent, the telephone calls never made? Did you think to yourself, "Gee, I wish I had been nicer to old Aunt Tillie..."? Well, the ISPUG membership seems to be universally suffering from an Aunt Tillie syndrome; we've received more mail (and more good information) in the past few weeks than ever did flow in during the course of a year--all of it prompted by our announcement last issue of the imminent death of the Gazette. Don't feel guilty about that; the reaction to demise is human and universal.

We summarize below some most useful information which so arrived; God Wot we'd had it a year ago! Before we do that, however, we publicly thank Terry Peterson, Joe Bostic, Steve Zeller, John Frost, Gary Ratliff, Reg Beck, Loch Rose and Stan Brockman, the stalwarts who contributed consistently to the Gazette and formed the technical foundation on which ISPUG was built. No man is an island; the editor of this rag merely provided a focus around which the real work was done, not only by the gentlemen above, but by all the others whose names have appeared in our columns for the past four years. To all, our gratitude; to all, Godspeed.

FROM HITHER, THITHER & YON

ADA Interfaces: Connecticut Microcomputers, 150 Pocono Road, Brookfield Connecticut 06804, (203) 775-4595, still makes a complete line of interfaces for serial and IEEE (and, we suspect for parallel) connections to 8032s and to SuperPET, primarily to interface printers in combination with disk drives. We have used their equipment for five years, and are most happy with it. Dan Kesler of 442 Pin Oak Drive, Sunnyvale CA 94086 reports that they still make all this gear. If you need an interface, get in touch with CM.

Repair at SKYLES: Skyles Electric Works (415) 965-1735, in Mountain View, CA, still repairs 8032s and SPETs and our disk drives, according to Dan. Call before you ship the gear.

Reg Beck Offers to Keep APL Group Going

See Reg's column this issue for details (and for his address).
If you're into APL and want to participate, call Reg or write him!

Bug in SuperPET SuperFORTH: John Spencer of 519 Elsie St., Edwardsville, IL, 62025, has found a bug in John Toebes' FORTH, in U* word. He says that because this is a machine-language primary used by several other words, the error could show up in routines that don't even use U* explicitly. If anyone is interested, John offers a screen that revamps the erring ML, a couple of bytes shorter than the original, so that no patch is required. Get in touch with John directly. We are sorry to say that John Toebes' SPET has been hors de combat for the last five months; it's out for repair, but nothing seems to get fixed...

Where to Get SPET Fixed...

Maintenance in Central U.S.: James A. Segler of Pekin IL writes that he "knows of a shop that maintains 25 or 30 SuperPETs in nursing homes and does good work on my two SPETS, drives, and printers. The service man's name is John; he really knows what he's doing when it comes to Commodore equipment." Try CBM Computer Center, PO Box 509, 1622 Fourth St., Peru IL 61354 (815) 223-0102.

More on Mid-Kansas Computer: James Segler also notes that he recently bought a SuperPET from Mid-Kansas (see p. 275, issue 10, Volume II for more data) for $246. It arrived well-packed, in mint shape and with a chip for Visicalc. James adds "I have since purchased other items from them by phone; have had no problem at all." If you wonder why we report this, see issue 10 (reference above) for a not-so-favorable report on this same firm. Also see the next item.

Character Generator Out? Ralph J. Redman of 2016 Elmwood Lane, Pueblo, CO, 81005 lost his character generator a few months back and had to search for a new one. He reports that Mid-Kansas Computers, PO Box 506, Newton Kansas 67114 (316) 283-0208, has these chips in stock for SuperPET and that he received his new one about three days after he ordered. They don't take plastic money, but will ship C.O.D. We note several previous reports on good repair work at this firm.

Fast Repair at Diamond Bar: Loren D. Felten of 21622 Stanwell St., Chatsworth CA 91311 reports that On Line Microcomputer, in Diamond Bar CA, is very good and fast at repairing SuperPET. Unfortunately, Loren gave us no phone or address, so get in touch with Loren for it--or use directory service.

KEYBOARDS!: Bill Cronkhite of 1250 Cresthaven Drive, Pasadena, CA 91105, says that his "neighborhood Commodore store was able to, very quickly, obtain a new keyboard for one of my SPETs. It's made by Mikita and has a much better touch than the original. I now have a spare and do the refurb on it and swap when one goes bad. I also got some small black stick-on rubber feet from Radio Shack, about .5-inch in diameter, for the inside of the metal cover of the keyboard, to replace the originals which fall out and roll under the filing cabinet..." Well, we don't have the address or phone number of that neighborhood store, but we do have Bill's. So ask him.

SPET BBS and Maintenance Help: Darrell J. Damon of 5728 - 83rd Street, Kenosha Wisconsin 53142 (414) 697-0622, writes, "if there is any interest among SPETters I'll run a bulletin board in this area, provided it gets used, dedicated to SPET questions, answers and software. Have people call or drop me a line." He also says he'd like to keep in touch with other SPETters in his area, adding that he is something of a hardware man and has managed to cure most of his SuperPET's ills. He says he'll be glad to extend what knowledge to anyone who needs help; he used to repair older PETs in his spare time and finds SPET little different.

Comment on This and That

Turbo PROLOG: John Seither III of 4242 Briars Road, Olney MD 20832, tells us that as long-time advocate of COBOL and other third-generation languages, he was a bit negative toward PROLOG (a fifth generation language). He reports his first reaction to Turbo PROLOG, the newest release by Borland International (famous for their Turbo PASCAL): "Having played with the language for a few days, I am forced to concede that its ease of programming, I/O access more characteristic of assembly language, and impressive abilities in the design of expert systems may eventually eclipse earlier generations of language in popularity and utility. In any event...watch for it in future releases [for other computers]."

Orphans, Orphans: W.F. Wright of Marble Falls, Texas, writes that "As the owner of an orphan SuperPET and an orphan Victor 9000, I feel that my collection now lacks only a 1936 Packard." C'mon, W.F.: you still don't own an Edsel!

Bodsworth Marches On! Don Littlefield of Arvada, Colorado say "I've served in two wars, flown the Alaskan bush, detonated nuclear devices...got out of Teheran before the Shah, worked with Yuppies, survived three assignments in New Jersey, and thought I had seen it all. Now what? The Gazette is going to fold! What is to become of Dickie Dumbjohn and Bodsworth...?" Both gentlemen, despite the lack of grey stuff betwixt their ears, immortally march on in the Amigan A&J, Don--

And we're afraid Don is right when he states "You and I and the other SPETters who enjoy languages are the vast minority of computer users. The market is driven by a need for appliance-like machines no more complex than the telephone. To do this, one needs...software packages which run at blinding speeds and do not require a 'socially demeaning' keyboard." Yup. Take the airliner to Peoria, but don't bother to learn to fly there yourself. This is exactly the reason we not long ago bid a tearful farewell to our beloved white scarf and goggles.

Last, Don notes you can rejuvenate ribbons with a shot of WD40 (on the revolving felt which keeps the ribbon inked). He enclosed a sample, which frankly bled pretty badly into the paper. There's an even easier way: buy a bottle of black stamp-pad ink, (with a tiny pouring spout) and re-ink the felt. Ribbons continue to print nicely until the fabric wears out.

Printer Output to Serial: Delton B. Richardson of Norcross, Georga says he has rigged up a letter quality printer to SPET's serial port, but couldn't get the printer to linefeed (it did a CR okay) when he tried to copy from disk to that printer. But--if he first got the file into RAM, and then said: PRINT SERIAL, both linefeed and CR output to printer okay.

DELPHI and TPUG: Nicholas Kaps of 378 Sheridan Court, Manteca, CA 95366, notes that TPUG on DELPHI (he's speaking of a telecom net) is a good contact point for the remaining SPET owners. If you want more information on how to get aboard, we suggest you get in touch with Nick.

Robert Davis

We regret to announce the recent death, from leukemia, of Bob Davis, our former associate editor in Pascal, who was plagued by the disease for a number of years. His widow, Doris W. Davis, 100 Darrow Drive, Pennington, N.J. 08534, has contributed all his SuperPET gear--except for a modem--to a nearby school. The modem, a Hayes Stack Smartmodem 1200, never used, and complete with all the manuals, is available for sale. Sorry, we don't have her number. Use directory service, or write.

For Sale or Wanted

SPET and Gear for Sale: David L. Briggs of 4318 East 60th St., Tulsa, Oklahoma 74135, (918) 492-9448 [home] or (918) 492-4440 [work] offers the following gear for sale at the first reasonable offer or plea. He isn't at all opposed to donation for a worthy cause: SPET, 8050 drive, 8022 printer, all manuals including COBOL, all issues of the Gazette. All equipment is in good condition.

Help on Graftrax Plus: Barry L. Walden of 1408 Forest St., Thunder Bay, Ontario, Canada P7C 2X2 has an Epson MX80 printer and wants to retrofit Graftrax Plus. He asks if it is possible, and if so where to get the parts? Drop him a line if you can help. We wouldn't recognize Graftrax if it walked up and bit us.

German Character Set Wanted: Tony Klinkert (PSC #1 - Box 4785, APO New York, 09633 (Residence Schiersteiner Str. 18, 6200 Wiesbaden, West Germany, 011-49-6121-84-1143 [home]) wants to upgrade his SPET to German. Needs information on where to find PaperClip 9000B and a character set ROM or an already-created German character set printer-file for 8300P (Diablo 630) printer. Write or call.

CLOSEOUT ON COM-MASTER FOR $25.00

ISPUG distributes COM-MASTER, a telecom program and terminal emulator which handles the ASCII or APL character sets, text or binary (PRG) files in upload or download, and lets you set any telecom parameter you care to. You may also save versions of the program, tailored as you want them, for specific applications. For more information, see list of SuperPET software in this issue.

Because of the discontinuance of the Gazette, Dan Jeffers (the author) is offering COM-MASTER for $25 until existing stocks of manuals run out (we have only 34 left in stock). Order from ISPUG. The last of the wine, the last of the lace, the last of the splendid Paris dresses...

Indexed by Courtesy of Marilyn

The index to Volume II of the Gazette, appended to this issue, was created by Marilyn Post of Dillon, Colorado, the only SPETter (apparently) who plays tennis and also is unaware of the old Army rule that you never, ever volunteer (a rule, we add, that the majority of ever-silent SPETters seem to know by heart...).

Stan Cook, a mathematician from British Columbia (now an Amigan) writes that "I feel quite guilty for not providing anything for the newsletter--it's just that I felt it would not measure up to all the great stuff from your other contributors." Well, Stan, here's a secret: Our job was to edit; the writer's job, to set down the facts--which often came in a jumble, written on soggy tablet, misspelt, littered with the ruins of English. Editors are consigned to this planet to convert such stuff into readable articles--and to enjoy the process!

INSTALLING AND USING THE HIGH-RES TECHNOLOGIES GRAPHICS BOARD

by
Nick Solimene
87 - 27 94th Street
Woodhaven, N.Y. 11421

[Ed. Last issue, we reprinted an article by Tom Stiff on a high resolution graphics board useful on SuperPET. Nick Solimene got a board early on. Following is his report on how he installed and used it.] The board works very well indeed and adds quite a lot to the performance of SPET. The display area of 1024 x 512 pixels is impressive. Unfortunately, SPET's screen provides a window to only 700 x 270 pixels. The full area can be seen, however, by panning the window over the entire display.

Documentation: The documentation is a sparse 11 pages, including the instructions on how to install the board and a demo disk. Installing it in my SuperPET (three boards) was not as simple as I would have desired. The board is plugged into the 40 pin socket on the lowest SP board and the cable normally attached there is plugged into the hi-res board.

Installation: The problem is that the ROM towers at UD11 and UD12 are too tall. The instructions suggest that you insert a 40 pin socket to raise the hi-res board so that it clears the ROMs. This was clearly inadequate on my machine.

In particular, one tower was much taller than the other. I rebuilt that tower by soldering a low profile header and socket together and resoldered the attached resistor and the wires from the UD11 switch to the new tower. [The use of heavy braided wire (#14 ?) is beyond me.] Now, both towers are about the same height and about as low as possible short of their removal and the consequent disabling of the UD11 and UD12 switches.

Even with two 40 pin sockets (albeit low-profile), the hi-res board rests on the ROMs and is is not firmly socketed. Moreover, the reduced space above the hi-res board, caused by the extra sockets, made it impossible to connect the CRT cable without first bending the connector pins from vertical to horizontal. Although the present installation works, it isn't satisfactory. As soon as I can get some taller spacers to raise the second board, I will use a third 40 pin socket so that the ROMs will be cleared and the board will be firmly socketed.

Comments on Hardware: The board seems well-constructed and has eight 64K bit chips for the graphics memory, some logic chips and the CRT controller. Display parameters are set by poking to the registers of the CRT controller. Graphics memory is accessed via a one-byte window after poking a two-byte address. The hardware doesn't provide for synchronizing the graphics display to the normal character display. This is done in software. You may display and scroll both characters and graphics independently. Either or both may be turned on or off.

There seem to be some differences in the CRT controller parameters used on the two sides of the SuperPET and the documentation for the 6502 side. I wish more information had been provided on the CRT controller. It seems to have an interlace mode; I wonder if using it would double the number of pixels displayed to 700x512. Will write to Hi-Res Technologies for more information.

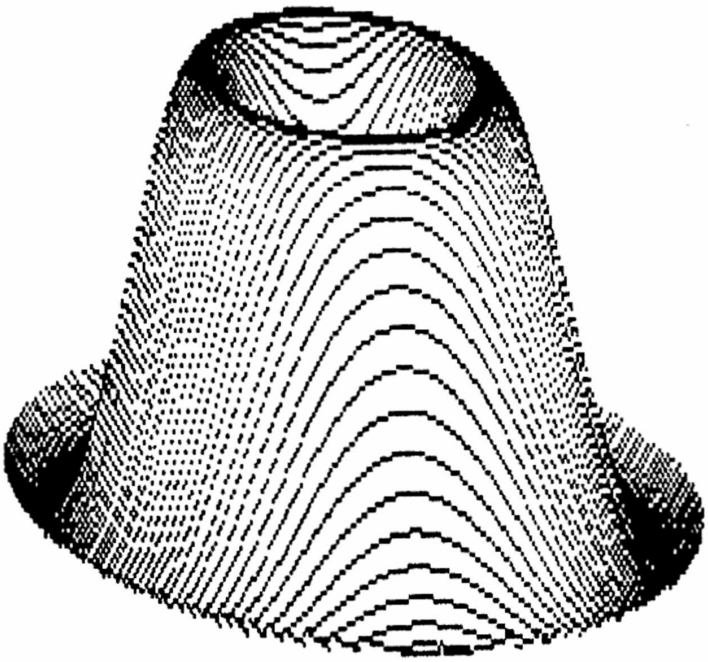
The Software: The software provided is rudimentary but sufficient to illustrate the capabilities of the hi-res board and how it may be programmed.

Some 6502 demos are provided. The ones I've looked at are in machine language with BASIC4 drivers. The main ML routine allows calls from BASIC4 using SYS 320xx to set window size and origin, fill screen (clears with 0), draw line between two points, reset parameters, adjust vertical or horizontal, or reverse field. The reverse seems to be missing in my copy or the instructions are wrong. The parameters are passed by using the locations of BASIC4 variables. It would have been nicer if these commands had been wedged into BASIC4. The source code isn't provided; the routines are partly described by giving BASIC4 equivalents using PEEKs and POKEs. There are some typos in the documentation; a critical OR is missing and a ')' was dropped. Looking at the 6809 source cleared this up.

For the 6809, there are some programs by Avy Moise. One group concerns the reconfiguration of OS/9 to use the graphics memory as a ram-disk. These I haven't looked at. There is also a graphics demo for use from the monitor, plus a demo in FORTRAN. The source code for the 6809 graphics demo is provided but it makes use of a macro named 'procedure' as well as one named 'call'. I have 'call' on one of the ISPUG disks but don't recall seeing 'procedure' anywhere previously. I'll try to get a copy. In any event, the .mod files are provided; you don't have to assemble to use demo.

Except for trying some of the demos, I have done my graphics work using HALGOL, because of its speed and because the large memory of the 68K Grande board make

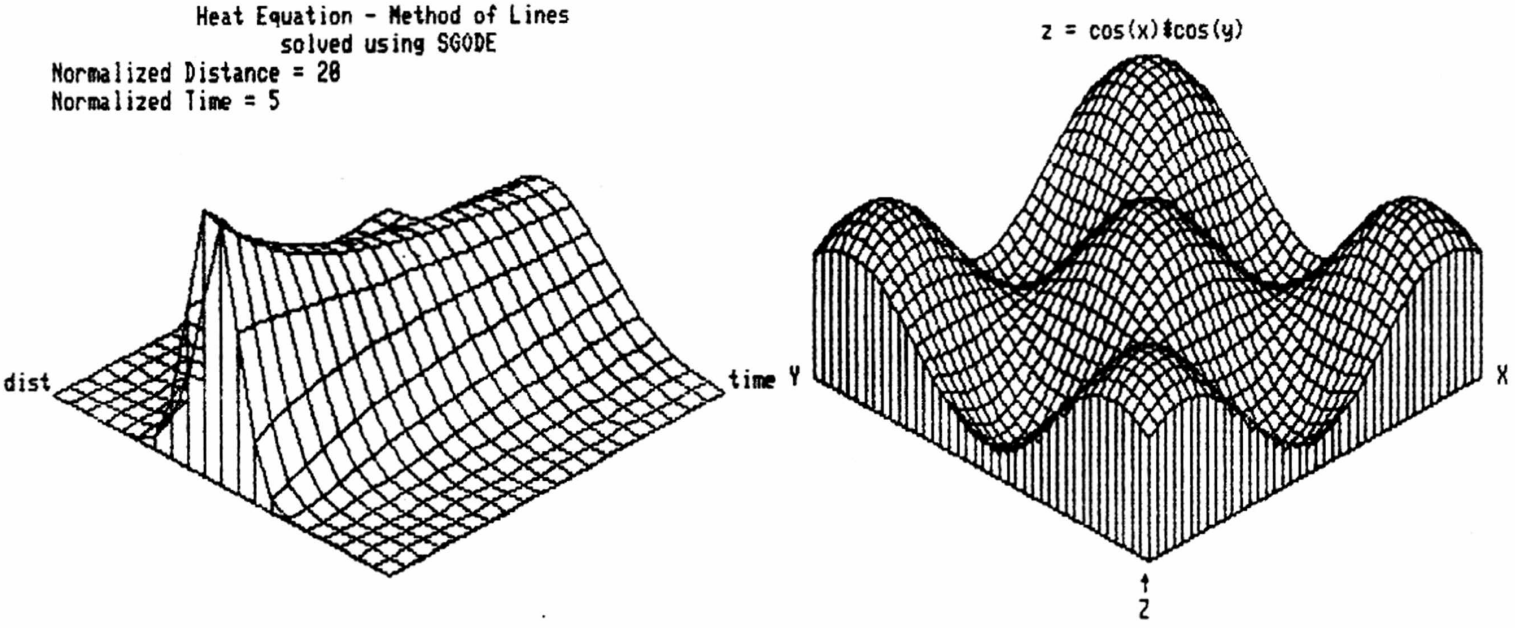
the graphics most useful. I've added nine commands to HALGOL to handle graphics, which wasn't as difficult as I had expected.



I converted the MTU hat program (see left) from an ancient ad by MTU to HALGOL and with some fiddling ran it in about 2.5 minutes; the original program was quite slow.

HALGOL does very well, and makes feasible display of 3D shapes of this variety (not necessarily figures of revolution) without having to wait forever.

The printer dump of the hat at left was done in BASIC4, using PEEKS and POKES of the graphics memory to build up data strings for my printer; the dumps are done sideways. Each graphics byte accounts for eight adjacent horizontal bits; dumping sideways therefore requires no bit manipulations. The dump rquires about six minutes. The demo disk has a ML dump for the 4022 printer, but it was far easier to use BASIC4 than to attempt to modify that program--especially without source available. Show below are two more plots as examples of what the board can do.



These were dumped directly from HALGOL by calling 6502 ML routines to manipulate the bits; run time is about two minutes instead of the six needed with BASIC4.

[Ed. If you're interested in the graphics board and require a tad of help, Nick says its okay to get in touch with him at the address above, but cautions that as he uses the graphics board, the DTACK GROUNDED board is needed as well as the Hi-Res Technology board, and that all his programs are written for modified HAL-GOL, now renamed as HBASIC.]

T H E A P L E X P R E S S by REG BECK

Box 16, Glen Drive, Fox Mountain, RR#2, Williams Lake, B.C., Canada V2G 2P2

In the last column, we looked at direct disk access in APL but didn't have room for an application. The following functions will read in any sector on the disk, and produce both a hex and character dump. Unprintable characters are replaced by periods. The function SWAP lets us toggle back and forth between ASCII and APL character sets as we look at the character dump of the sector. SWAP 1 gives us ASCII, SWAP 2 yields APL. In Vol. II, No. 4, p. 104 we offered two functions for this purpose. These functions, APL and TXT, used the poke codes given in the Waterloo System Overview Manual and worked well except for differences in the punctuations between the character set we obtained and the normal ASCII set. We then read through back issues of the Gazette and found a note (Vol. I, No. 9, p. 115) which advised to SYS to location 45194 to get correct results. The arguments are 1 for ASCII and 2 for APL. READ was defined in the last column.

```
      DISPLAY 'DTH'
DTH:(16 48)⍴,((⍉('0123456789ABCDEF')[⎕IO+(2⍴16)⊤⍵]),' ')
      DISPLAY 'CHARTODEC'
CHARTODEC:-⎕IO-⎕AV⍳⍵
      DISPLAY 'REMOVE'
REMOVE:¯1↓⍵,⍵[(⍵∈⎕AV[1+(⍳13),127,255])/⍳⍴⍵]←'.':(⎕IO←1)=0:''
      DISPLAY 'SECTOR'
SECTOR:(DTH CHARTODEC READ ⍵),'|',(16 16)⍴REMOVE READ ⍵
      DISPLAY 'SWAP'
SWAP:⍵ ⎕SYS 45194
⍝ITS NICE TO HAVE THESE IN DIRECT FUNCTION DEFINITION!  ⎕IO WAS MADE
⍝LOCAL ONLY IN 'REMOVE'.
```

We could have defined one function to convert directly from characters to hex but separated the two operations in the two functions DTH and CHARTODEC. REMOVE gets rid of the unprintable characters. All the functions are put together in SECTOR. The syntax used is SECTOR 18 1 to display sector 1 of track 18, for instance.

We have gone to some trouble in these functions to make them independent of the index origin. REMOVE was modified to make QUAD IO a local variable. More on this later in the column. In REMOVE, the conditional form was used to ensure an index origin of 1. The condition is looked at first which sets QUAD IO equal to 1. The condition of 1 = 0 can never be true so the alternate case will never result. A little trickery was used to make REMOVE result-returning since the form of replacement used (replacement of the unprintable characters with periods) modifies the vector but doesn't return it.

* * *

I recently came across an article by Howard Rotenberg in Commander, (Jan., 1983) titled Radix-50: Pack & Unpack. He discussed an old method used on the PDP-8 computer (8 instructions, 4K memory) for packing code. It converts three bytes into two bytes for a memory saving of one third. With only 4K available on the

PDP-8 the appeal of this method is obvious. Forty characters can be used, which include the alphabet, the numbers, space and three other characters. The algorithm splits text into groups of three characters. Each character is replaced by its code (0 to 39) and each group of three codes is packed into a single number using base 40. All such numbers are less than 65535 and can therefore be represented by 2 binary bytes. This scheme is a natural for APL's encode and decode primitives.

```
PACK:40⊥ALF⍳(3,(1÷3)×⍴M)⍴M←(3×⌈(1÷3+⎕IO←0)×⍴ω)↑ω
UNPACK:,ALF[3⍴40+⎕IO←0)⊤ω]
CHECK:ALF[40|ALF⍳¯1↓ω,▼⎕IO←0]
ALF←' ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.,;'

⍝⎕IO←0 APPEARS IN THE FUNCTIONS IN DIRECT DEFINITION AS A LOCAL VARIABLE.
⍝CHECK TAKES TEXT AS A RIGHT ARGUMENT. IF THE TEXT CONTAINS CHARACTERS NOT
⍝IN ALF THEY ARE CHANGED TO SPACES.
⍝THE RIGHTMOST PART OF PACK, UP TO THE ASSIGNMENT ARROW, ENSURES THAT THE
⍝NUMBER OF CHARACTERS IN THE TEXT IS DIVISIBLE BY 3. THE REMAINDER OF PACK
⍝FORMS THE TEXT INTO A 3 ROW ARRAY, CONVERTS EACH CHARACTER INTO ITS INDEX
⍝IN ALF (THE NUMBER OF ITS POSITION IN ALF) AND THEN ENCODES EACH COLUMN.
⍝UNPACK DECODES THE BASE 40 NUMBERS BACK INTO TEXT.
```

The idiom used in CHECK is interesting. The residue function is used to compress any disallowed characters out of the text. Details follow:

```
      ⎕IO←0
      TEXT←'THIS≠>>>≥IS-÷+×IT'
      ALF⍳TEXT       ⍝FINDS INDICES OF TEXT IN ALF.
20 8 9 19 40 40 40 40 9 19 40 40 40 40 9 20
      40|ALF⍳TEXT    ⍝FINDS REMAINDERS WHEN THESE INDICES ARE DIVIDED BY 40.
20 8 9 19 0 0 0 0 9 19 0 0 0 0 9 20
      ALF[40|ALF⍳TEXT]   ⍝THE COMPLETE IDIOM.
THIS    IS    IT
⍝IN THE LAST STEP CHARACTERS WERE SELECTED FROM ALF ACCORDING TO THE NEW
⍝INDICES. THIS METHOD WORKS BECAUSE ANY CHARACTER NOT IN ALF IS GIVEN AN
⍝INDEX OF 40, ONE GREATER THAN THE LARGEST INDEX.
```

We wanted QUAD IO to be a local variable in these functions so that the index origin in the workspace would keep its original value. To do this we modified the direct function compiler to accept the QUAD symbol as part of a local variable. (Another way to do this would be to modify the function by using the DEL editor after compilation.)

```
⍝TO ADD MORE SYMBOLS TO THOSE WHICH WILL BE ACCEPTED BY THE DIRECT
⍝FUNCTION DEFINITION COMPILER AS PARTS OF LOCAL VARIABLES MODIFY LINE 4
⍝IN LV9 BY ADDING ⎕ TO THE LIST. ANY VARIABLES HAVING A ⎕ IN THEM WILL
⍝NOW BE PLACED IN THE HEADER OF THE COMPILED FUNCTION AND WILL BE LOCAL.
      ∇LV9[⎕]∇
[  0]    Z ← LV9 S ;ASS;VAR;A;GET;E;L
[  1]   ⍝PUT LOCAL VARIABLES IN HEADER FOR DDEF
[  2]   Z←''
[  3]   →0×⍳1=+/ASS←S∈S[S⍳':']←'←'
[  4]   VAR←S∈'ABCDEFGHIJKLMNOPQRSTUVWXYZ_⎕'    ⍝⎕ ADDED TO THE LIST HERE.
[  5]   GET←VAR∧A∈(VAR∧1⌽ASS)/A←+\VAR<1↓V̄AR,0
```

```
[  6]   Z←(4≠⎕NC Z)⌿Z←'←' B9K (GET∨ASS)/S
[  7]   Z←(0,1↓(A⍳A)=⍳ρA←(Z∧.=⍉Z)⌈.×⍳1↑ρZ)⌿Z
[  8]   Z←(' '≠Z)/Z←,';',Z
```

Lets put all this together with some functions to assist in creating binary files.

```
CHECK 'THIS◇WILL◇REPLACE□DISALLOWED∆CHARACTERS⍳WITH÷SPACES.***'
THIS WILL REPLACE DISALLOWED CHARACTERS WITH SPACES.
TEXT←'NOW****IS THE TIME FOR ALL GOOD MEN, ETC.'
PACKED←PACK CHECK TEXT
⍝PACKED IS A VECTOR OF BASE 40 NUMBERS OBTAINED FROM TEXT.
PACKED
23215 24375 37324 200 13 245 614 15158 30400 45 32500 13283 8037 280
⍝IF WE UNPACK PACKED WE GET THE ORIGINAL TEXT (MINUS THE ASTERISKS WHICH
⍝WERE REMOVED BY CHECK).
UNPACK PACKED
NOW    IS THE TIME FOR ALL GOOD MEN, ETC.
⍝THE NEXT FUNCTION, HILO, CONVERTS THE BASE 40 NUMBERS TO HIGH-BYTE, LOW-
⍝BYTE FORM.
HILO:,⍉256 256⊤⍵
HILO PACKED          ⍝CONVERTS PACKED TO HIGH-BYTE, LOW-BYTE FORM.
90 175 95 55 145 204 0 200 0 13 0 245 2 102 59 54 118 192 0 45 126 244 51 227
  31 101 1 24
⍝THE FUNCTION, FIX, CONVERTS THE HIGH-BYTE, LOW-BYTE NUMBERS BACK TO BASE 40
⍝FORM.
FIX:256⊥⍉(((1↑ρC)÷2),2)ρC←⍵∘.+,0
FIX HILO PACKED
23215 24375 37324 200 13 245 614 15158 30400 45 32500 13283 8037 280
⍝WE CAN DO THE WHOLE OPERATION IN BOTH DIRECTIONS:
UNPACK FIX HILO PACK CHECK TEXT
NOW    IS THE TIME FOR ALL GOOD MEN, ETC.
```

We can use atomic vectors to save a binary file on disk. Each base 40 number created by PACK is first changed to high-byte, low-byte form and then used to index QUAD AV.

```
⍝TO SAVE TO DISK AS A BINARY FILE:
      R←HILO PACK CHECK TEXT
      'EX1' □CREATE 1
      □AV[□IO+R] □WRITE 1
      □UNTIE 1
⍝TO RETRIEVE FROM DISK:
      'EX1' □TIE 1
      Z←□READ 1
      □UNTIE 1
      UNPACK FIX (□AV⍳Z)-□IO
NOW    IS THE TIME FOR ALL GOOD MEN, ETC.
```

These functions and examples are presented more as an exercise in using encode and decode than as a practical text compression method, though the method works for that purpose and does save disk space. You can learn more about encode and decode by dissecting these functions and playing around with the parts.

* * *

Use the following method to deal with tie numbers.

```
⍝⎕NUMS HOLDS A VECTOR OF ALL CURRENTLY ACTIVE TIE-NUMBERS.
      ⎕UNTIE ⎕NUMS
⍝WILL UNTIE ALL TIED FILES. THIS IS HANDY IF YOU HAVE ERRORED OUT
⍝OF A FUNCTION AND DON'T KNOW WHICH FILE NUMBERS WERE USED. IT CAN
⍝ALSO BE USED IN A FUNCTION TO FREE UP ALL TIE NUMBERS.
      ∇CENTRE[⎕]∇
[  0]    LINE CENTRE TEXT ;X
[  1]   X←(⍴TEXT)+⌈0.5×80-⍴TEXT
[  2]   (¯80↑X↑TEXT) ⎕POKE 32768+(80×LINE-1)+ 1 2⍴  0 79
```

CENTRE is useful for centering text on the screen when giving instructions or presenting a menu. LINE refers to the screen line on which the text is to be centered. 2 CENTRE 'MENU' would centre MENU on the second screen line.

* * *

We looked briefly at sorting in a previous column. While there is no actual sort primitive function in microAPL, the primitive functions grade-up and grade-down are used and result in terse expressions for alpha sorts. The APL literature has much to say about sorting but usually moves on quickly to applications. Most of the messy and sordid details of the actual sort are left to the machine language routines working invisibly under APL. This is REAL high level language! An explanation follows the APL examples.

```
      X[⍋X;]  ⍝THE VECTOR SORT X[⍋X] WAS EXTENDED TO SORTING ARRAYS
              ⍝IN MICROAPL 1.1.
      Y←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      DISPLAY 'SORT'
SORT:ω[⍋(1+⍴Y)⊥Y⍳⍒ω;]
      DISPLAY 'RSORT'
RSORT:(⍴ω)⍴(,ω)[A[⍋(,⍒(⍴⍴ω)⍴⍳1↑⍴ω)[A←⍋,ω]]]
```

X	X[⍋X;]	SORT X	RSORT X
SMITH	APPLE	APPLE	HIMST
ZULUS	JONES	JONES	LSUUZ
JONES	SMITH	SMITH	EJNOS
APPLE	ZULUS	ZULUS	AELPP

In earlier APLs (and microAPL 1.0, I believe, though I have never used it) the vector sort was not extended to arrays. Methods such as the one used in SORT and several others (found in APL, An Interactive Approach by Gilman and Rose) were available. RSORT sorts the rows of an array separately. The table in the above example shows each sort applied to the array, X.

This is the last edition of THE APL EXPRESS, I'm sorry to say. I would like to thank all readers of the column, especially those who wrote with comments and suggestions and also to thank the Editor, Dick Barnes, for his patience with my sometimes obscure code and text. SuperPET APL users are invited to contact me to exchange information at any time. Please write directly to me and send all the particulars. If there are enough of you we might continue THE APL EXPRESS on a separate basis. Please send Canadian postage (or a couple of quarters) and a self-addressed envelope.

ISPUG DISKS WILL CONTINUE TO BE AVAILABLE!

As we said last issue, all ISPUG disks will continue to be available--so long as the insulation fails to rot off our SuperPETs and our disk drives continue to function. All back issues of the Gazette likewise will be available. Send your requests to PO Box 411, Hatteras, N.C., 27943. What we make on disks will be used to support the **Amigan A&J.**

"HOOKS" IN THE SUPERPET SYSTEM ROMS or, How and Where to Revise ROM Routines

With the help of Terry Peterson, Joe Bostic and Brad Bjorndahl, we summarize in this article how you may write and use your own assembly language routines to modify or replace the routines of SuperPET's operating system, which are, of course, embedded in ROMs. Waterloo provided "hooks" in some ROM routines which enable you to intercept a call to that routine and to direct the call instead to your own program.

The Flag and Patch Address Area: In 6809 mode, go into the monitor; examine the code between $0580 and $0600. You'll almost always find that it is full of zeroes although memory on both sides holds much non-zero data. At boot, or whenever you switch to 6502 and back, the area is cleared to all zeroes by a system routine at $E77A, which is shown at left.

```
LEAS    -2,s
LDD     #$0580
LOOP
  STD   ,S
  SUBD  #$0600
  QUIF  CC
  CLR   [,S]
  LDD   ,S
  ADDD  #$0001
ENDLOOP
LEAS    2,S
```

For convenience, we'll name memory between $0580 and $0600 the "table"; from $0580 through $05C0 it is reserved for "flags". If any value other than 0 is found in a "flag", it indicates that a patch exists. The value of the flag (assume it is 06 for this discussion) is an offset value--an offset from $05C0 (the start of the patch address area in the table). The address at that offset ($05C0 + 6 for this example) should be the address of the substitute routine which is to be executed in place of the system routine. We later show one way to store the proper values at the proper locations.

How the Hooks are Implemented Obviously, if there are patch flags, the system routines which can be patched must look to the patch flag area in the table before they execute any library routine. All such "patchable" routines therefore call PATCH_, a routine at $E76B. We show it below:

PATCH_ : $E76B Patch Sensing/Switch Routine

```
TFR     D,X             ; Enter with the flag offset in D Accumulator.
LDB     $0580,X         ; Load flag (if any) at that offset + $0580.
IF NE                   ; If the flag isn't ZERO
  TFR D,X               ; Transfer the flag offset value to X,
  JMP [$05C0,X]         ; and jump to the routine whose address is at
ENDIF                   ; $05C0 plus the flag offset.
RTS                     ; This routine assumes A register is always clear.
```

Which Routines Can be Patched? The next question, of course, is which of the ROM routines are 'patchable'? Using SPMON1, we searched all ROMs and got the following references in the B through E ROMs (there are no calls to $E76B in A or F ROMs). The addresses below are the actual calls to $E76B, not the start of

the routines themselves (a few are opcodes; we didn't sort them out). As you can see, a significant number of routines may be patched. If any system routine does not produce the results you want, check it for a call to $E76B to see if it can be patched.

BCD8 BD17 BD59 BE75 BF01 BFA1 BFF8 C000 C078 C0D4 C0E3 C140 C24B C2AD C2E0 C4DA
C534 C671 C6CE C7FF C84E C8D1 C912 C96B C9DE CAA1 CAF2 CBA0 CC1B CCEC CD2C CD62
CF01 CF4F D0B1 D122 D4DA D4F6 D504 D50F D51C D6AD D71A E1F4 E2F4

Patching a Routine Let's assume we don't like the TGETCHR_ routine at $D4D2 (John Toebes describes the flaw in this routine in II, 70), and want to substitute a patch. We show a disassembly of library TGETCHR_ below. Note what goes on at the double asterisks ** beside the code. First, Accumulator D is loaded with an offset of $27; we then JSR to the PATCH_ routine at $E76B, which is printed above.

```
PSHS  D
LEAS  -$01,S
LDD   #$0027 ;**
JSR   $E76B  ;**
JSR   $D6A7
STB   ,S
CMPB  #$0D
IF EQ
  LDB  #$01
  STB  [01,S]
ENDIF
CLRA
LDB   ,S
LEAS  $03,S
RTS
```

Every routine which references $E76B and which we've examined loads a different offset into the flag area of the table. We expect there may be some duplicate locations, but in essence each routine has its own flag address, established by offset from the base flag location of $0580.

Here, the offset is $27. If we want to patch TGETCHR_, then, we first must enter a flag at ($0580 + $27 =) $05A7 in the table.

Second, the byte we place at $05A7 must be a non-zero offset to the location of the address of our 'patch', at least one byte above $05C0. Let's assume the patch routine is located at $7F00 in main user memory. If we then store that address at $05C1 (one byte above the start of the patch address table), we must set the patch flag at $05A7 to $01. We do exactly this in the program printed at the end of this article.

It is entirely possible to patch a system routine to do your own work, and then to call the system routine to finish as usual, provided: (1) that you re-enter the system routine after the call to PATCH_ (to call the whole routine would be an infinite recursion). In TGETCHR_, above, you might patch and then again enter TGETCHR_ at any point from $D4DC onward; (2) if you take pains to make sure that the stack pointer is properly adjusted. This must always be done.

As Joe Bostic warns, you had best know thoroughly the system routine you propose to patch. We demonstrate below with a patch for TGETCHR_. We install our own routine, let it run, then cancel it by calling system routine $E77A (see start of this article), which zeroes all entries in the patch table. We thus restore the normal operation of TGETCHR_ at the end of a very brief program.

Assemble and link this program for origin at $7F00. Load "patch.mod" in the monitor; >GO it at $7F1D (see label 'start', below). The program pauses until you press a key and then press <RETURN>. It then will print at the center of the screen whatever character you entered immediately prior to a CR. You can also prove the patch ran by examining the byte EORFLAG at $7F37, which will hold $04 instead of the $01 normally stored by TGETCHR_. The program is a merely a short demo; except for clearing the EORFLAG each call, which TGETCHR_ doesn't, it is not useful.

```
;patch.asm  A demo of patching a system routine, TGETCHR_.
XDEF start, newget, eorflag
tgetchr_ EQU $D4D2

; The next section is a new TGETCHR_. It begins immediately after the call to
; PATCH_ ($E76B) in the original TGETCHR_. Note that the RTS in PATCH_ is never
; executed; instead the program JMPs to this patch. We therefore must adjust
; the stack pointer by two bytes for the not-executed RTS. All changes to the
; TGETCHR_ routine are asterisked.

newget    LEAS  $02,S           ; * Adjust stack for JMP from PATCH_.
          JSR   $D6A7
          CLR   eorflag         ; * Clear the flag which indicates a CR.
          CMPB  #$0D            ; Do we have a CR?
          IF EQ                 ; Yes,
            LDB   #$04          ;   * Flag it with a $04 (not 01).
            STB   [$01,S]       ;     Store value at label EORFLAG.
          ELSE                  ; * Not a CR;
            STB   ,S            ;   * Stack character entered.
          ENDIF
          CLRA                  ; No longer needed, but we left it in.
          LDB   ,S              ; Load B with GOT char.
          STB   $83E8           ; * Print character in mid-screen.
          LEAS  $03,S
          RTS

; The main program begins below. We set the patch flag and store the address of
; the patch routine. Then we call TGETCHR_. At the end, we zero out the patch
; table and resume normal system operations.

start     LDB   #$01            ; Load offset from $05C0 for address of new routine.
          STB   $0580+$27       ; Store the offset of 01 as flag.
          LDD   #$7F00          ; Load address of the "patch"; then
          STD   $05C1           ; store it where PATCH_ will look.
                                ; Now, call TGETCHR_.
          LOOP
            LDD #eorflag        ; P1 the address of the EORFLAG for TGETCHR_.
            JSR tgetchr_        ; Go to the patched routine.
            TST eorflag         ; If a CR was entered, quit.
          UNTIL NE
          JSR   $E77A           ; Zero out the patch table.
          SWI

eorflag   RMB 1
          END
```

It's interesting to note that the subroutine at $D6A7 (see the line right after label 'newget', above) also calls PATCH_--but with an offset of $04 into the patch table. If you trace all the subroutines called by the program above, you find a number of calls to PATCH_--but each has a different offset. We suspect it'd be wise to check out all such calls to PATCH_ in any substantial routine, on the off chance that two or more might have duplicate offsets.

An Exercise in EXECUTE: From Roger Bassaber, our faithful correspondent way down in the Indian Ocean, we got a note about the batch file capability of BEDIT to sort a list. If you're up to a game as well as an intellectual exercise, make the EXECUTE file at the left, which is of the search/do form; it says to find lines beginning with A, B, or C throughout the file, and to move them (A first) to the end of file. File this little batch file to disk as "test". Then clear screen and enter the list of names at left, with one Baker, one Charlie, and one Able. Then EXECUTE file "test". Aha! Is it fully sorted alphabetically? If not, why not?

```
*/%^A/ move $
*/%^B/ move $
*/%^C/ move $
```

```
Baker
Baker
Charlie
Charlie
Able
Able
```

Then try two of Mr. Baker, Mr. Charlie, etc. When you're well puzzled by that, change the MOVE command to ECHO... If you work at this a bit, you'll end up with a batch file which will sort a list of names alphabetically. Have fun.

A SECOND WAY TO PASS PARMS TO AND FROM MICROBASIC The technique in the article last issue passes short parameters to and from mBASIC; long ones require a different method. Dr. J. G. Cordes needed such a method because he had to pass very long floating point values back from an extended precision routine. He found that the general-purpose routine FINDVAR, written by Associate Editor Loch Rose, best served that need. This article explains the routine and how to use it.

FINDVAR is a machine language (ML) subroutine that locates a specified mBASIC variable in RAM. It will find integer, floating-point, and string variables, but cannot handle subscripted variables (array or matrix values). To use FINDVAR, you must define a parameter to be passed within mBASIC, either as a string, integer, or real value. FINDVAR itself must be written to know the name of that variable. We give some examples below.

1. Integer variables: Suppose you call a ML routine with a SYS. Before you do it, you store an integer value in language in the integer variable "param%". You want the ML program to use this integer in its computations. To locate this integer from your ML routine, you include the same name--'param%'--as a defined string in your ML program, as shown at left. As with all ML strings, you end it with a null byte. To find the location and the value of "param%", your routine must load the address of the defined string "param%" in D register and then call FINDVAR (see left). FINDVAR will then return the address of the value of "param%" as defined by mBASIC. This address returns in D register (if FINDVAR fails to find the variable, it returns 0 in D register). Knowing the address of the integer value, you may employ it as you wish in your ML routine.

```
intname fcc "param%"
        fcb 0
```

```
LDD  #intname
     jsr findvar
```

As Gary Ratliff said in Vol. I, p. 226 of the Gazette, the value of an integer is stored in two bytes following its name. FINDVAR points at the first of these bytes. You load D register with the value (see left), and then may manipulate it as you wish. Let us suppose now that you have used the variable and have calculated a new integer value. You want to pass it back to mBASIC under the same variable name of "param%". You find the location of "param%" in mBASIC just as before, but this time you put the value in the two bytes following its name in mBASIC's variable

```
tfr      d,x
ldd      ,x
```

```
                table, as we show at left, where "value" is the value of the new
tfr  d,x        integer variable. When you return to language from the SYS, the
ldd  value      value of "param%" will be that calculated in your ML routine. It
std  ,x         is very easy to pass or to return integer variables this way.
```

We emphasize that the mBASIC variable to which you wish to return a value must exist (if only as a valid dummy, with at least the length of the returned variable) BEFORE you SYS your ML routine.

2. Floating Point Variables: With one minor variation, the procedure is identical for reals. You must end the name of a FP variable in your ML routine with neither a "$" nor a "%", so the routine will know it is supposed to look for a FP variable (see left). You call FINDVAR just as you did for an integer (left, below). FINDVAR will return in D register the address of the byte following the variable name, or 0 if it is not found. Remember that floating point values are stored in five bytes following the variable name, in external FP format (see John Toebes' article in II, 5, p. 131).

```
varname  fcc  "float"
         fcb   0


ldd  #varname
jsr  findvar
```

3. String Variables: Strings are handled much the same way, but again with one exception. Strings themselves are not stored in the bytes following their names in the variable table. Instead, the two bytes following their names contain the address of the string itself. So, instead of manipulating data in those bytes, we manipulate an address.

```
strname fcc  "stringer$"
        fcb   0


ldd  #strname ; Find string
jsr  findvar  ; address.

tfr d,y       ; Look at
ldx ,y        ; two bytes.
```

We load the first bytes at the address into the X register (see left, below), check to see if the value in X is zero. This occurs if, perchance, you goofed and nulled "stringer$" in mBASIC. If X is not null, you have a good string, but it will not be terminated by a null. Instead, the two bytes preceding its address contain its length. You'll need this to stop reading the string at the proper place. Get the length into the D register and use it as necessary to stop reading at the right place (as at left). If you propose, in contrast, to stuff a new string where a dummy string was, be sure to not only store that new string at the right address, but to define the new length properly in the two preceding bytes, or mBASIC won't be able to read the string properly.

```
ldd  -2,x
```

Loch's ML routine, FINDVAR, follows:

```
;mainfind - mainline subroutine, i.e. resets top of memory

MemEnd_   equ       $22
Service_  equ       $32

main      ldd       #$7dff          ;Reset top of memory to make room
          std       MemEnd_         ;   for program
          clr       Service_        ;Return to Waterloo menu
          rts
          end

;Findvar - returns address of byte following a given mBASIC variable
```

```
;     Parameter:  D, address where string representing name of mBASIC
;                 variable (string, integer, or floating point) is stored

          xdef findvar,okflag
          xref length_,equal_

endprog   equ       $0044          ;contains address of end of mBASIC program
endvar    equ       $0046          ;end of mBASIC variables

findvar   std       straddr        ;save address of string
          tfr       d,x
          loop
            ldb       ,x+
          until eq
          leax      -2,x           ;adjust pointer to last byte of variable name
          ldb       ,x             ;load last byte of variable name
          pshs      x              ;save pointer
          guess
            cmpb      #'$          ;is last byte a '$'?
            quif      ne
            pshs      b            ;variable is a string, store '$' on stack
            ldb       #%00100000   ;used later to examine string variables only
            pshs      b
            clr       ,x           ;clear byte containing '$'
          admit
            cmpb      #'%          ;is last byte a '%'?
            quif      ne
            pshs      b            ;var. is an integer, store '%' on stack
            ldb       #%01000000     ;used to examine integer variables only
            pshs      b
            clr       ,x           ;clear byte containing '%'
          admit
            clrb                   ;variable is floating point
            pshs      b
            pshs      b            ;store two 0's on stack
          endguess
          ldd       straddr
          jsr       length_        ;find length of variable name
          cmpd      #0
          beq       nogood         ;if 0, end sbr
          cmpd      #31
          bhi       nogood         ;if >31 characters long, illegal name, end sbr
          stb       strlen         ;store string length
          ldx       endprog
          leax      1,x            ;adjust pointer to start of variables
          loop
            loop
              cmpx      endvar
              bhs       nogood     ;if past end of variables, give up
              ldb       ,x+        ;load next character in memory
              bsr       alphabet   ;is character alphabetic?
            until     ne           ;quit if alphabetic
            leax      -1,x         ;take back last autoincrement of X
            ldb       -1,x         ;load char. preceding alphabetic char.
```

```
          cmpb    #$20          ;is it a space char.?
          if      eq            ;if so
            ldb     -2,x        ;load character preceding space char.
            tfr     b,a         ;make a copy of B in register A
            andb    #%00011110  ;calculate length of variable name
          else
            tfr     b,a         ;make a copy of B in register A
            andb    #%00011111  ;calculate length of variable name
          endif
          anda    #%11100000    ;mask off the 5 LSB's of A
          cmpa    ,s            ;is it the type of variable we're seeking?
          bne     nomatch       ;no, we'll skip it
          cmpb    strlen        ;is length of name same as of one we want?
          bne     nomatch       ;no, skip
          pshs    d,x           ;save registers
          clra
          pshs    d             ;push length of name onto stack
          pshs    x             ;push address of name's start onto stack
          ldd     straddr       ;address of name of variable we want
          jsr     equal_        ;are the two names the same?
          leas    4,s           ;adjust stack ('leas' does not alter flags)
          puls    d,x           ;restore registers (neither does 'puls')
          bne     found         ;TRUE, we've found the variable's name
nomatch   leax    b,x           ;skip variable name
          leax    3,x           ;skip three bytes following name
          tsta
          if      eq            ;if A=0, variable is floating point
            leax    3,x         ;need to skip 3 extra spaces
          endif
        endloop

nogood    clr     okflag        ;failure, clear flag
          leas    1,s           ;get rid of comparison byte
          ldb     ,s+           ;load byte that ended variable name
          puls    y             ;address where that byte was
          if      ne            ;if byte wasn't 0
            stb     ,y          ;restore that byte
          endif
          ldd     #0            ;unsuccessful, return 0
          rts

found     lda     #255
          sta     okflag        ;success, set flag
          leax    b,x           ;address of byte at end of variable name
          leas    1,s           ;get rid of comparison byte
          ldb     ,s+           ;load byte that ended variable name
          puls    y             ;former address of that byte
          if      ne            ;if byte wasn't 0
            stb     ,y          ;restore that byte
          endif
          pshs    x
          ldd     ,s++          ;place address of byte after end of var. in D
          rts
```

```
okflag    fcb       0             ;0 if no luck, set to 255 else
strlen    rmb       1
straddr   rmb       2


;alphabet - returns TRUE if char. in B is alphabetic (including '_'), else FALSE
;                       (affects only D, flags)

alphabet  cmpb      #$7a          ;z
          bhi       notalpha
          cmpb      #$41          ;A
          blo       notalpha
          cmpb      #$5a          ;Z
          bls       ok
          cmpb      #$61          ;a
          bhs       ok
          cmpb      #$5f          ;_
          beq       ok
notalpha  ldd       #0
          rts

ok        ldd       #1
          rts
          end
```

RELATIVE FREQUENCY OF INITIAL LETTERS

We have found the tables below of value when creating and sizing files for words and names. Note the considerable difference between the frequency of initial letters as used in names and the frequency as used in words. Both tables are based on 1.0 as the frequency of the letter "q". Does anyone have a clue as to why names starting with "B", "S", "C" and "W" dominate the table of names? Are the hormones of mating and procreation somehow stimulated by the sound of them? Hmmm. Do you have a better theory? It explains the low esteem we accord to any quitters, quacks, quadroons, quagmires, quarrels, guibbles, quirks or quislings; and, perhaps, why we know by instict that Iago is a villain, the Id primitive and vicious, all idealists impractical (sic!), idols are not to be worshipped, idiots are ignorant or impudent, and incest ignoble. Why are words which begin with "q" and "i" so quaint or so incendiary? Why is swift ever so much faster than quick, and frozen so much colder than icy?

Initial Letter in U.S. Last Names
(From Telephone Books)

a	20.0	m	20.0
b	56.7	n	11.7
c	41.7	o	11.7
d	23.3	p	28.3
e	16.7	q	1.0
f	17.7	r	23.3
g	23.3	s	55.0
h	40.0	t	20.0
i	2.0	u	5.0
j	20.0	v	5.0
k	11.0	w	43.3

Inital Letter in English Words
(From Webster's Collegiate)

a	10.5	m	9.2
b	8.5	n	3.3
c	15.5	o	4.0
d	8.2	p	14.7
e	6.2	q	1.0
f	7.2	r	7.8
g	5.3	s	20.0
h	6.5	t	10.0
i	6.6	u	2.7
j	1.5	v	3.5
k	1.5	wx	5.3

```
1   20.0    xyz    1.7                    1    5.8     yz    1.2
          Total = 518.4                              Total = 166.0
```

Example Division of Alphabet into Rough Quarters for Both Tables:

Names				Words			
518/4.0 = 129.5 occurences/division				166.0/4 = 41.5 occurrences/division			
a-d	141.7	l-r	116.0	a-d	42.7	m-r	40.0
e-k	130.7	s-z	130.0	e-l	40.6	s-z	42.7

BASIC ACCOUNTING
by Fred Foldvary
1920 Cedar St.
Berkeley, CA 94709

Most accounting programs, such as Delton Richardson's "Home Accounting" program, are designed to read your data, calculate balances, and print reports. My BASIC ACCOUNTING is not an accounting program, but a method using the Basic language. Instead of inputting the numbers into a file, in BASIC ACCOUNTING (B.A. from now on) you enter the numbers as Basic statements. One could say that in B.A., the "medium is the message".

The sample program 'accounting:ba' is not meant to be used itself, but to illustrate the accounting method, which you may adapt as you wish. The beauty of the method is that you can do any simple--and perhaps even complex--accounting work with it--your personal expenses, income tax, or business accounting. It is very flexible.

Line 1000 holds the name of the program. I have several B.A. programs for various purposes. The :ba suffix means the program is a SAVED file. The 'save' which prefixes the name lets me LIST line 1000, move the curser to the line, delete the line number and hit RETURN. Voila, the program is saved! You may, of course, substitute STORE or add a version number or change the drive...

On line 1010, I can change the date; line 1030 asks for two inputs: (1) 't' or 'p' for output to terminal or printer; (2) the 'account' if the program has more than one. This specific program, for example, accounts for 1) a magazine called 'Topical Time' (abbreviated tt) which I edit, 2) personal expenses (per), 3) interest and dividends earned (which I designate by 'b' for IRS schedule 'B'). I can also designate 'all' for all accounts. So you enter t or p followed by a comma and one of the accounts. Note that you run the program only to calculate totals and write reports--not to enter data!

On line 1060, change IEEE4 as necessary for your printer; line 1080 prints the report heading; line 1110 initializes totals. In this simple example, I only have income and expenses, but you might initialize whatever totals you need, perhaps calling a procedure to do so.

Lines 1130-1230 call the various accounts. Then lines 1250-1270 print total income, expenses, and difference as net savings. Again, adapt this to your own needs, perhaps in a procedure if it is long. In 1250-1260, I round off net savings to the nearest cent since BASIC's arithmetic may leave a sum of 3.9999 for 4. There may be a simpler way to do this.

Following line 1300 are the actual accounts, each a procedure. Your data goes into Basic statements of the form: nnnn variable_name = variable_name + value ! comment: (date, check #, item).

For Topical Time magazine, I follow the terms and numbers in IRS schedule C, for this makes it easy to fill out the schedule at tax time. Gross receipts, for example, go on C's line 1, so I use the variable name: gross_receipl. This we add into C line 5, gross income, which I have in gross_incom5. Note: at this time, C line 5 only has an addition from line 1, but if I later need to have an entry for schedule C line 4b, I would simply change BASIC line 1340 to something like: gross_incom5 = gross_recepl + other_income4b. The beauty of BASIC ACCOUNTING is that you can modify the data and calculations as you get new figures.

At line 1350, supplies25 corresponds to schedule C, line 25, supplies. Following the amount, the rest of the line is a comment on that particular expense, such as the date, check number, merchant, and description. On line 1370, supplies25 is incremented by the new value, commented with a date and a note that I bought envelopes and tape for cash at Coop Hardware. At line 1350, I used my MasterCard (MC); at 1360, a Homestead check number 123 (Hom123).

Want to find where a check number was used? Search for /Hom123/. In line 1390, I calculate the sum of the expenses from only two variables, which correspond to Schedule C, line 32. The net profit (C line 33) is calculated and printed in lines 1400-1410; cumulative income and expense are done in lines 1420-1430. The line renumbering feature in microBasic makes it easy to add statements.

Personal expenses are done similarly, except that these are not tax items, so there is no form to follow. I just use various categories of expenses (e.g. car_expense and goods). Note that some values are divided by two--useful when you share expenses with another person.

That's all there is to it. Every expense, income, and other accounting number is added or stored into some variable name. These keep a running total; at the end you do whatever calculations are needed and print out the totals. When you want to see the totals, you just run the program. Everything is visible right in the program, which doubles as your data file. Your basic accounting can be done on SuperPET with BASIC ACCOUNTING.

```
 1000 ! save 'disk/0.accounting:ba'
 1010 ! saved Feb. 28, 1986
 1020 !                Parameters
 1030 print "Enter output file: t or p, account: tt, per, b, all"
 1040 input out_file$,which_account$
 1050 if out_file$ = "t" then out_file$="terminal"
 1060 if out_file$ = "p" then out_file$="ieee4"
 1070 open #3,out_file$,output
 1080 print #3," 1986 Accounting"
 1090 print #3
 1100
 1110 income=0 : expense=0
 1120
 1130 if which_account$ = 'tt'
 1140    call topical_time
 1150 elseif which_account$ = 'b'
 1160    call interest_dividends
 1170 elseif which_account$ = 'all'
 1180    call topical_time
 1190    call personal
```

```
1200    call interest_dividends
1210 elseif which_account$ = 'per'
1220    call personal
1230 endif
1240 !
1250 net_savings%= (income-expense+.001)*100
1260 net_savings = net_savings%/100
 1270  print  #3,"Income=";income;"    Expenses=";expense;"    Net=";net_savings
1280 close #3
1290 stop
1300 !
1310 ! ***************** Topical Time ****************************
1320 proc topical_time
1330 gross_recepl=                 700 ! ATA Jan
1340 gross_incom5= gross_recepl
1350 supplies25  =              22.58 ! Ja 10; MC; Radston's; envelopes
1360 telephon28  =              40.05 ! Ja 16; Homl23
 1370  supplies25   = supplies25 +   2.50 ! Fe 11; cash; Coop Hard; envs, tape
1380 telephon28  = telephon28 + 34.76 ! Fe 20; Homl25
1390 total32     = supplies25 + telephon28
1400 net_profit33= gross_incom5 - total32
1410 print #3,"TT net profit=";net_profit33
1420 income= income + net_profit33
1430 expense= expense + total32
1440 endproc
1450 ! ******************Interest/Dividends************************=
1460 proc interest_dividends
1470 interest=         53.40! Jan Homestead
1480 interest=interest+33.24! Dec Pac Coast
1490 interest=interest+34.33! Jan Pac Coast
1500 interest=interest+ 4.66! Dec Crocker
1510 print #3,"Total interest=";interest
1520 income= income + interest
1530 endproc
1540 ! ****************** Personal ********************************=
1550 proc personal
1560 car_expense =             28/2 ! Ja  6; Embl04; CSAA Auto Club
1570 goods       =              7.50! Ja  6; Embl05; NatSoc; wallet
1580 subscription=             24/2 ! Ja 17; Cro784; Tribune
1590 goods       =     goods + 34.08! Ja 31; MC; Attache Case
1600 service     =             40/2 ! Fe  4; Cro770; 44 Sweep; chimney
1610 house       =            476/2 ! Jan 1
1620 house       =     house +476/2 ! Feb 1
1630 car_expense =car_expense +59.89! Fe 11; MC; tires & smog
 1640  total_personal=  subscription + car_expense + goods + service + house
1650 print #3,"Personal expenses=";total_personal
1660 expense= expense +total_personal
1670 endproc
```

Disk Doctor Will Soon Open for Practice

Associate Editor Stan Brockman has been laboring for almost a year to write an easy-to-use, menu-driven program by means of which non-gurus can recover bad disks, amend data, read tracks and sectors--in short,

can doctor their disks as they darn well please. He says the finished version will be ready by September 1, in 8050 or 4040 format. If you want a copy, order from Stan, at 11715 West 33rd Place, Wheat Ridge, Colorado 80033. Send $10. Written in assembly, but runs for tyros. How do we know? We've run three beta test versions...

RESTORE TO A LABEL

by Bob Wherritt
2913 North Vassar
Wichita, Kansas 67220

There seems to be no way to RESTORE (i.e., reset a READ of a data list to a given point) to either a line number or a label in mBasic as in some Basics. Yet, mBasic has hidden talents. If a READ variable is a string variable, mBASIC treats all data items as strings. So we can stick a <data-label> at the start of a sublist we wish to read, set up a loop to READ it, and discard all items until the <data-label> is READ. Then we can READ the items we want. To stop at the right place, we can count, or better yet, we may put an <end-label> at the end of the sublist.

```
restore
loop
  read discard$
until discard$ = "superglues"
loop
  read realitem$
  if realitem$ = "endsuperglues" then quit
  process realitem$
endloop
data poorglue, paste, $1.29, 243
data superglues
data stickem, grabbit, holdit, leggo!
data endsuperglues
data etc
```

The program outline at left demonstrates the idea, without any frills. The first loop READS up to and including the label and discards it, leaving the first desired data item read for access. Note that any data item preceding the item or list we want can be numeric or a string. When an <end-label> is READ, the program leaves the processing loop.

It seems a good idea to put the labels on a separate line so they are easy to see, read and edit.

The idea for the RESTORRE to LABEL is my own, but the neat idea for an end-label to allow for sublists of indefinite length is Dick Barnes'.

For one or two sublists we can put together routines similar to the one shown above. In long programs in which several sublists exist, and especially if they may be revised, it is wise to set up special procedures with error-trapping, as shown in the program listing below.

```
1000 !  RESTORE TO LABEL — demo to print out sublists of data items
...  !  written by:   Robert C. Wherritt
...  !
1280 ! This is a procedure to restore the data list to a preassigned label.
1320 ! And a second procedure to print out to screen and printer
1360 ! the data items listed under that label. Error trapping is included.
1400 ! Also included is a test program to show how it works!
1440
1480 proc restorre(category$)
1520   on eof
1560     st = 2
1600     resume next
1640   endon
1680   st = 0 : empty = 0
1720   restore
```

One calls this procedure after stuffing the data-label in category$. These three lines trap the end-of-data error.

This line RESTOREs the whole data

```
1760   loop                                  list; then this loop READs and dis-
1800     read datta$                         cards all data items up thru the
1840     if st = 2 then quit                 data-label.
1880   until datta$ = category$
1920   if st = 2                             If end-of-data is reached first,
1960     st = 0 : empty = 1                  then it sets the flag empty.
2000   endif
2040 endproc
2080
2120 proc printt (category$)
2160   enderror$ = "This data sublist has no end-label."
2200   call restorre(category$)
2240   quitflag = 0 : endflag = 0                  This is a data-processing
2280   print : print category$;":   ";             procedure, which would
2320   if empty                                    surely be different in
2360     print "No such data label!"               each application.  This
2400   else                                        one can serve as a model.
2440     open #5, "ieee4", output
2480     print #5 : print #5, category$;":  ";
2520     loop                                      After fixing error messages,
2560       st = 0                                  this procedure calls re-
2600       read what$                              storre.  It prints the label
2640       if st = 2 then quit                     and if there is no such
2680       if idx(what$, "end")                    data label, it shouts this
2720         endflag = 1                                          and ends.
2760         if what$ = "end"+category$ then quitflag = 1         If there
2800         if quitflag then quit                                is such a
2840         print enderror$ : print               label, it reads and prints
2880       endif                                   out all data items which
2920       if quitflag or endflag then quit        follow that label, until
2960       print what$,                            it reaches a data item
3000       print #5, what$,              containing the string "end".
3040     endloop                         If it doesn't find an end-label,
3080     if st = 2                       then it will print out all of
3120       st = 0                        the remaining data items and end
3160       print : ! print #5            with an error message. (If the
3200       print enderror$               programmer doesn't put end-labels
3240     elseif st = 0                   where they belong, what a mess!)
3280       print                         When it finds an end-label, it
3320       print #5                      quits READing and checks to see
3360     endif                           whether the end-label matches
3400     close #5                        the last data-label.  If so, it
3440   endif                             ends; otherwise, it issues an
3480 endproc                             error message before ending.
3520
3560 data cats                           This is a label-line.
3600 data Owl-Eye, Don Quixote
3640 data Prissy, Hopto
3680 data endcats                        This is an end-label-line.
3720 data amount                         Label-line.
3760 data 25, 43.7
3800 data endamount                      End-label-line.
3840 data dogs                           Etc.
3880 data Dorian, Rufus, Mona
```

```
3920 data enddogs
3960
4000 call printt ("dogs")
4040 call printt ("cats")
4080 call printt ("numbers")
4120 call printt("amount")
4160 call printt ("figures")
4200 call printt ("dogs")
```

THE APPROXIMATE PRINTOUT
(Starred lines have no hard copy.)

```
dogs: Dorian   Rufus   Mona
cats: Owl-Eye  Don Quixote  Prissy  Hopto
numbers:  No such data label!   **
amount:  25   43.7
figures:  No such data label!   **
dogs:  Dorian  Rufus   Mona
```

[Ed. When we scan Fred Foldvary's Basic Accounting (elsewhere this issue) and this article, we stroke our beard and wonder if the DATA from BA might be even easier to handle as data statements mit labels...]

CURRENT LIST OF SUPERPET SOFTWARE

Because we've had a lot of calls of late for a current list of ISPUG disks and other SuperPET software which is now available, we print below a summary of that software. We list only that material designed to run in SuperPET, (either in 6502 using SuperPET's 64K of bank-switched memory), or in 6809 mode using the 6809 microprocessor. A great deal of software runs on the 6502 side, and is covered in depth by COMPUTE!, by RUN, by TPUG and by the Midnight Review/Paper. If disk are ISPUG, write to ISPUG, PO Box 411, Hatteras, N.C., 27943 for 4040 or 8050 format. If we recommend you get copies from author, do so. Please, please state the format! About half of you forget... All prices in U.S. dollars.

PAPERCLIP Word-Processing Program. Available at your Commodore dealer, or write: Batteries Included, 186 Queen Street West, Toronto, Ontario Canada M5V 1Z1. Get version 9000A or 9000B, designed specifically for SuperPET. $150.00. WordPro compatible. 9000A allows up to 816 lines of text in one screen file; 9000B, with some other improvements, provides 718 lines. Overall, an excellent and versatile program. A program for the 6502 side is available, but has limited capacity, since it does not use the upper 64K of SuperPET memory. Should your dealer be uncertain which of three versions of PAPERCLIP to give you, ask for the EXPANDED version. The disk directory will show either 9000A or 9000B. Reviewed in issue 11, Vol. I, p. 173.

ISPUG MASTER TELECOM Telecommunications programs written especially for SPET, in machine language for speed. You need know nothing of ML to use them. The programs let you to communicate with another SuperPET, other micros, a minicomputer, or a mainframe. One is very simple, for SPET to SPET. The second may be reconfigured as you wish, and each version saved to disk to do a specific job. Works with CompuServe and other networks. Full instructions in plain English are on disk, plus full data on how to configure the RS-232 port and on SPET telecom operations in 6809. With the 2031 disk drive you'll drop a few characters because of the drive's limited buffer. Programs work well with 4040, 8050, or 8250. Full upload/download capability for SEQ but not for PRG files.

For 6502, find a complete package of programs and instruction, designed particularly for use with Punter Bulletin Boards. The programs are rewritten to handle the 6551 ACIA chip in SuperPET, to avoid having to jumper connection at the RS-232 port. Full instructions on how to use the BB programs are on disk. $25 U.S. from ISPUG in either 4040 or 8050. $15 U.S. for 6809 only or 6502 only.

TELECOM FROM THE LANGUAGES This disk contains three assembly-language programs which enable telecommunication at 1200 baud or less from SuperPET's languages. The BASICOM package works with microBASIC; APLCOM with APL. You do not need to understand assembly language to use the routines; you will have to write your own routine, in your language, to do what you want to do. Examples of how to do this in both mBASIC and APL are on this disk, with instructions. In addition, there's a third package, UNICOM, which provides a machine-language package to tie into other SuperPET languages, with an explanation of how to do it. See p. 246, issue 14, Vol. I for more info. $10 U.S. from ISPUG in 4040 or 8050.

COM-MASTER A menu-guided, interrupt-driven, fully buffered professional telecommunications package for SuperPET. It emulates the Lear-Siegler ADM-3A terminal. Full upload/download capability on SEQ and PRG files; control of all telecom parameters. Save any version, after tailoring to your needs from menu, to disk. Allows use of command files, called from menu, to automatically configure the program. ISPUG's APL Editor says it's the "best APL terminal emulator available." By using reverse field, it marks underbarred APL characters, and allows full interchange of APL files with I. P. Sharp. Not limited to APL; handles files from all SuperPET languages and general telecom. Manual and disk. State format. Distributed by ISPUG; tech support by Quality Data Services. $50 U.S from ISPUG.

PETCOM Another menu-driven, fully buffered, interrupt-driven telecom package, well written and with a very good manual (it covers telecom in SPET in general as well as its own program). Will run while mBASIC, mFORTRAN, mEDIT, or mPASCAL are being used. DOS commands may be employed from the program. Handles PRG and SEQ files, controls all telecom parameters. Not for APL. Advantage lies in use with languages listed. Designed primarily for employment with minis and mainframes, though useful with commercial networks and micro to micro. Call or write: Ph.D. Associates, Suite 200, Kinsman Bldg., Downsview, Ontario, Canada M3J 1P3, 416 667 3808. Last price quoted was $99 Canadian. Excellent program. Special APL version available. Write for information.

SUPERPET BULLETIN BOARD Program lets you set up as a System Operator (SYSOP), and run your own bulletin board, which is accessible to all other computers, from PC's to Radio Shack Color Computers. For 8050 format, write ISPUG. For 4040, write: Paul V. Matzke, PO Box 574, Madison, Wisconsin 53701, the author. $6 U.S. Program runs in 6809, in microBASIC, and by report does well at 300 baud.

HOME ACCOUNTING A professionally-done way to handle your personal finances; it uses relative files (transparent to you and easy to operate), menu-driven, with a tutorial on disk. Available in 8050 format only from its author, Delton B. Richardson, 4299 Old Bridge Lane, Norcross, Georgia, 30092. Also includes programs which will generate attractive bar-graphs very flexibly; includes tutorial on these. Price: $15 U.S.

MICROPIP A Peripheral Interchange Program (PIP) which handles files better than any program we've seen. It also saves part of memory as a machine-language load module, or compares two files either in text or program format, and tells you where the differences are and what they are. You can get selective directories of specific filenames (tele* will give you a directory of all files starting with the prefix 'tele'); you can sort, copy, or rename a

bunch of files with one command; in all commands you can optionally log results to your printer. If you teach, handle a disk library or use assembly language much, this is a jewel. Available from: WATCOM Product, 415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2. Price: $75. Manual and disk.

PRINTING APL CHARACTER SET Articles, workspaces, and data files which show you how to print the APL Character Set to the Epson MX, FX or Commodore 8023 printers, and also show you how to design your own set and use it. Sets will work on printers listed, but the approach should work on any dot-matrix printer which accepts column definitions of its characters (the vertical slices of the dot-pattern). Specifically for SuperPET. On disk also are a number of utility programs which alphabetize disk directories, print files from from main menu, provide a disk directory from menu, etc. $10 U.S. from ISPUG.

ISPUG UTILITY DISK A compendium of the best and most useful programs written by ISPUG; includes two extended monitors, SPMON and EXMON, more powerful than MICROMON for 6502. Includes assembly language screen dumps suited for any printers, alphanumeric sorts, programs to print any directory or any SEQ file from main menu, programs to recover bad disks, a series of programs which will generate very flexible bar-graphs from any printer, screen dumps to disk useful everywhere (including the monitor), programs to recover a lost language/program after inadvertent exit, to set time and date from main menu, and a rewritten version of the microEDITOR which is fast as scat, plus more. Fills 2 4040 disks. For 8050, $10 U.S.; for two disks in 4040, $16 U.S. from ISPUG.

ISPUG UTILITY DISK II A second disk of major new programs plus some some useful utilities. Contains BEDIT, the new and very powerful Editor, with selective directories (and sorted directories), MOVE, ECHO, and a batch file capability. Included is CALC in several versions, which provides a powerful calculator/adding machine/conversion calculator in SPET, plus BEDCALC, which puts the same capability into the editor, BEDIT. Also on disk are programs to perform DOS work at main menu and copy or delete files at a keypress. All programs are well documented; most are supported by tutorials. Included also are all Waterloo patches for mBASIC and mFORTRAN through May, 1985, and some useful small utilities. $10 U.S. from ISPUG in 8050; two 4040 disks for $16.

STARTER-PAK A tutorial disk and manual which will save you about two months of pain, wrath, and indignation when you first crank up SuperPET. Tells you how to handle disk files, your printer, construct filenames, use the DOS commands, get and print directories, write programs which input/output to disk, or output to printer. I/O examples in all languages but COBOL (COBOL Waterloo tutorials do this). Most new owners don't realize all languages but APL are written in and edited from the microEDITOR, adapated to the language, so the material concentrates on mastery of the mED. Workspaces to handle all APL I/O are included on disk. Also on disk are a number of utility programs, such as screen dumps to printer/disk, a directory sort (alphabetical), a program to get disk directories from menu, one to change disk drive device numbers, etc. Includes most of the things the manuals should have told you and didn't. ISPUG. 30-page manual and disk. Order 4040 or 8050 from ISPUG. $15 U.S.

SuperPET SuperGRIT Teaches touch-typing the Commodore keyboard; runs in 6809 mode. Covers both the main keyboard and the keypad in 25 lessons. Takes "grit" to make up your mind to start, and "grit" to finish. $10 U.S. in 8050 or 4040 format from ISPUG.

SuperPET SuperFORTH A full implementation of FORTH for the 6809 side of SPET. Files are fully compatible with and accessible from the 6809 languages. Fully documented on disk. You will need a beginning book in FORTH in addition to the disk, such as "Starting Forth", by Leo Brodie. $35 U.S. on one 8050 disk or two 4040s from ISPUG.

SUPERPET GRAPHICS PROGRAM, 8023 PRINTER A 6502 machine language/BASIC 4.0 program creates high-res graphics on the Commodore 8023 printer. Menu-driven, with a fine set of instructions and a tutorial on disk. Fills one 8050 disk; not available in 4040 format. The upper 64K of SuperPET memory stores the graphic images you create; the program outputs them to the 8023, and is capable of the finest resolution which the printer can produce. You need know nothing about machine language to use the program, which is friendly and not difficult to learn. Source code on disk if you care to revise it. $10 U.S. from ISPUG. Distributed as user-supported software (support available from the author), who would appreciate a contribution if you like the program.

SUPERPET GRAPHICS PROGRAM, THINKJET PRINTER Same as above, except that it is designed for the Hewlett-Packard Thinkjet Printer. Not available in 4040. $10 U.S. from ISPUG, and distributed as freeware.

SUPERPET BAR-GRAPH APPLICATION DISK Uses the routines from the disks above to produce bar-graphs from either the 8023 printer or the ThinkJet. This is one application of the many which may be developed from the general routines which are provided on the above graphics disks. See Vol. II, p. 241 for a sample of output. Can do up to 41 bars at one pass. You enter the data, the program gener- ates the bars and prints them automatically. Menu-driven. In either 4040 or 8050 format. $10 U.S. from ISPUG.

CORDES EXTENDED-PRECISION FLOATING POINT PACKAGE Includes four machine language extended-precision routines which will provide, respectively, 10, 18, 40, and 50 digits of accuracy when it performs arithmetic operations on decimal num- bers. You need not know or understand machine language to use the routines, which are driven by programs in microBASIC. Variables are entered as microBASIC variables, and results return as named language variables. The necessary micro- BASIC driver programs are also on disk, together with a program named "calc:bu" which can use any of them to produce, as a screen calculator, the results of any arithmetic operation on big numbers, to any of the precisions listed above. The program will iterate any operation as many times as you wish. Full source code is included for assembly language programmers who want to extend it.

SUPERPET SUPERFORTH An implementation by John Toebes, VIII and Dr. Hal Levin, especially for the 6809 side of SuperPET. Comes on two 4040 disks or one 8050, so state format! Requires background in Forth (recommend "Thinking Forth" and "Start- ing Forth" by Leo Brodie). Includes an introduction and glossary plus ten manual sections with an index to the manuals. Employs both user memory (32K) and the 64K of bank-switched memory. See details, Vol. II, No. 7, p. 182. $35.00 from ISPUG, of which $25 goes to the authors.

APL EDA DISK For details, see pp. 89 and 110, Vol. I, Gazette. Implements John Tukey's (Princeton) work on Exploratory Data Analysis (EDA). For

the journeyman APL user; contains some powerful tools. $10 U.S. from ISPUG.

APL ANSCOMBE DISK Frances Anscombe, a Yale professor of statistics, issued a book titled "Computing in Statistical Science through APL," which has been supplemented by a disk for SuperPET, called the "Anscombe Disk." Available also from TPUG as T7. $10 U.S. in 8050 or 4040 from ISPUG. Disk and book recommended for APL journeymen in statistics.

APL COMPILER For details see p. 100, Vol. II, Gazette. Contains a direct definition compiler for APL, written by Ted Edwards, which enables you to define what is to be done; the compiler writes the APL code. Disk also contains a tutorial on how to use the compiler; you may page through it, forward or backward; the tutorial is interactive. $10 U.S. in 4040 or 8050 from ISPUG.

SPMON1 A final version of this powerful extended monitor for the 6809 side of SuperPET, with complete instructions. For details, see p. 116, Vol. II, Gazette. Contains an assembler, a powerful disassembler, commands to trace, compare, load or save, transfer, and display code. You may divert output either to disk or printer. Also provides two modes, QuickStep and Walk, by which you may examine code, step by step, as it executes, with full register displays. $30 U.S. from Editor in 4040 or 8050 format, of which $20 goes to the author.

HOSTCM A Waterloo program to directly interface SuperPET to mainframes or minis, available for IBM VM/CMS and DEC RSTS/E operating systems. See review, Vol. I, p. 102, Gazette, (No. 8) for features, price, and source.

MICROEDITOR DISASSEMBLY/REASSEMBLY For the competent assembly-language programmer, done by John Toebes of ISPUG with an assist from Bill Dutfield of TPUG. Disk containing all source files, and a reassembly/relinked which runs. For any one who wants to see or rework the code. Matches Waterloo's V1.1 mED. $6 U.S. in either 4040 or 8050 format from ISPUG. This is NOT speedy V1.3 as supplied on the ISPUG Utility Disk, nor as powerful as BEDIT.

OP SYSTEM DISASSEMBLY Again for the competent assembly-language programmer. Partially commented disassembly of SuperPET's operating system, by John Toebes with an assist from several other members. Fills three 4040 disks, for $12 U.S. In 8050, one disk for $6 U.S. from ISPUG.

::

NOTE: ISPUG disks with instructions or tutorials normally sell for $10. Those priced at $6 contain few instructions. Those which sell for more than $10.00 require that ISPUG pay royalties to the author(s).

A SIMPLE WAY TO GET TIME DIFFERENCES
(Gee, Delta_T)

In mBASIC, though not in all languages, SPET does not report time in seconds and jiffies (1/60ths of a second) when you call intrinsic function 'time'. Instead, you get seconds from startup. If you want to time anything more accurately, you have to peek the 'jiffies' at $163. By luck, we found out that the jiffies are reported by the peek as numeric values, so you can add them directly to 'time' if you first divide by 60 to obtain fractional seconds. Both seconds and jiffies are quantified; if you happen to call for a time difference between the beginning and end of any precisely repeatable event, you find that the "rollover" to the next jiffy and the next second will sometimes report times that are off by one full second.

We couldn't figure a way to evade this problem, and happened to send a program using our 'timecheck' to Frank Brewster, who arrived at a splendid solution--and with a new 'timecheck' which, in 99.7% of the calls to it, will report elapsed time accurate to 1/60th of a second. In the remaining .3% of cases, the accuracy is within + or - .02 seconds of elapsed time. (In defining accuracy, we do not take into account fractions of jiffies, so don't snarl at us, please...). The results above came from 1000 iterations.

If you want to compare the relative performance of different approaches to a problem in which speed is essential, 'timecheck' is most valuable. You call it before an event starts, and call it again when it ends. The time for the procedure itself to run is zeroed out (in 99.7% of all calls). The procedure reports the differential time, or 'delta_t', for each event to occur. You can call it as often as you wish, for it toggles automatically to handle each call. Example:

```
270 proc timecheck
280   corr=0
290   if peek(355)=59
300     corr=.0173
310   endif
320   t=(time+peek(355)/60)
330   if toggle%
340     delta_t=ip((t-tim-corr)*100+.5)/100
350   else
360     tim = t+.0793
370   endif
380   toggle%=1-toggle%
390 endproc
```

```
call timecheck
...event
call timecheck
print delta_t
```

If called with no intervening event, from language only (not SHIFT/RUN from mED), it reports its own time to run as zero, 99.7% of the times used. Thanks, Frank.

Refunds of Membership Fees

Each of you will receive a check from ISPUG sometime in late August or September, for unexpired memberships. We've ordered the checks, must write a program to scan the membership lists, calculated what's owed, and print the checks. We can't possibly do that until after the next issue of the Amigan A&J is sent to printer (gee, it's astounding how little sleep you get when you publish two journals...). When your check arrives, stand silent in the night; you should hear us snoring...

Prices, back copies, Vol. I (Postpaid), $ U.S. : Vol. I, No. 1 **not** available.

No. 2: $1.25	No. 5: $1.25	No. 8: $2.50	No. 11: $3.50	No. 14: $3.75
No. 3: $1.25	No. 6: $3.75	No. 9: $2.75	No. 12: $3.50	No. 15: $3.75
No. 4: $1.25	No. 7: $2.50	No. 10:$2.50	No. 13: $3.75	Set: $36.00

---Volume II---

Numbers 1 thru 11: $3.75 each. Set: $38.00 Postpaid, U.S. and Canada.

Add 30% to prices above for additional postage if outside of North America. This price includes postage! All back issues will continue to be available!

==========

Send all orders for disks and back issues to: ISPUG, PO Box 411, Hatteras, N.C., 27943, USA. Make checks to ISPUG.

SCHOOLS!: Send check with Purchase Order. We do not voucher or send bills!

This journal is published by the **International SuperPET Users Group** (ISPUG), a non-profit association; purpose, interchange of useful data. Offices at PO Box 411, Hatteras, N.C. 27943. As of this issue, the journal will no longer be published. Back issues and ISPUG disks may be obtained from ISPUG, PO Box 411, at Hatteras, N.C. 27943. SuperPET is a trademark of Commodore Business Machines, Inc.; WordPro, that of Professional Software, Inc. Send appropriate postpaid reply envelopes with inquiries. Canadians: enclose Canadian dimes or quarters for postage.

Table of Contents, Issue 11, Volume II

SuperPET Gazette
PO Box 411
Hatteras, N.C. 27943
U.S.A.

First Class Mail
in U.S. and Canada.
Air Mail Overseas.