SuperPET Gazette

We thank all the members who wrote or called to see if ye editor had blown away during the recent hurricane, the center of which passed about ten miles east of our office. Except for blowing down a large part of the Mickey Mouse Power & Light Co.'s new power line (Mickey Mouse P&L can't do anything right), the hurricane did less damage than a hard winter nor'easter. When we heard that the eye of the storm had come that close to home, we visualized chaos and desolation; when we drove home a day later, expecting ruin, we found nary a shingle shifted, nary a board out of place. The hurricane was a pussy cat. After dancing a thankful jig, we drove to the Post Office and found a letter from Commodore approving our request for a development Amiga and its documentation. Day of despair, day of jubilee!

ISPUG's disk library and files wouldn't have been wiped out if the hurricane had been a tiger. We took SuperPET, two disk drives, and our files with us when we evacuated, and spent a useful day in a motel working on this issue. SuperPET is transportable, more or (groan!) less...

Amiga and Expiration

So long as we have enough members to pay for the operation, we'll continue to publish the Gazette, which, as you may guess, will soon carry a comment or two on the Amiga, though we'll still concentrate on SuperPET. If your address label is redmarked, bless us with a check and send your address label or a copy so we can identify the culprit. Don't fill out the application form if you send the mailing label; just check the RENEW block.

If any of you have interest in a new publication dedicated to the Amiga, please drop us a note. We haven't made up our mind, but we'd like to be encouraged...

ANOTHER LANGUAGE FOR SUPERPET: TOEBES and LEVIN RELEASE FORTH

Our resident genius, John Toebes, and his co-worker, Dr. Hal Levin, have released SuperPET SuperFORTH, a full implementation of the language. Before you rush out to buy it, you should get the book "Starting Forth," by Leo Brodie; John recommends you also read "Thinking Forth," by the same author. The language comes on one 8050 disk or two 4040s, and holds, in addition to the language files, an introduction, a glossary, and ten manual sections, plus an index to the manuals. You'll need the books recommended; the implementation assumes a beginning familiarity with the terms, constructions, and techniques of Forth.

ISPUG acts as the distributor for the language; technical support is furnished to purchasers by Levin and Toebes. For such support, write: John Toebes, VIII, 120H Northington Place, Cary, N.C. 27511.

As the authors note, many Forth implementations break an entire disk drive into 1024-byte sections, which is fine for Forth, but means that no other types of files can co-exist on that disk. Toebes and Levin have made their FORTH DISK compatible with the Commodore DOS. It can be read not only from Forth but from any Waterloo editor or language. The FORTH DISK is implemented as a REL file under the DOS; this usage is transparent when you are in Forth (you don't have to fuss with REL file commands). You can BACKUP any disk or file and may copy any programs or files with the Commodore DOS. The system is compatible with all Commodore drives. In addition, you may store and retrieve any normal 6809 files on the same disk. You can even write a program in one of the Waterloo languages which modifies a Forth file.

The price of the language is $35 U.S., of which $25 is remitted to the authors; the remainder goes to ISPUG for copying and distribution. The disks are not copy-protected. For your copy, write Editor, PO Box 411, Hatteras, N.C. 27943. Please state disk format (4040 or 8050).

ONCE OVER LIGHTLY Miscellany Louis Mittelman, Jr. of Gordonsville, VA, writes us that the Smithsonian Institution, National Museum of American History, was most happy to accept his old PET 2000 and other related items for its collection. Anyone who thinks he has a rare computer item and wants to donate should get in touch with Dr. Uta C. Merzbach, Curator, Division of Mathematics. Sorry, Louis didn't give us a phone number or address.

BITS & BYTES HAS A BUG Our erstwhile Associate Editor, Gary Ratliff, has a sick computer, and is unable to provide a Bits & Bytes column for this issue. We shipped him a set of schematics and a prayer, but his SPET is still bugged.

HAS CANADA FLOATED AWAY? Gee, we haven't been in Canada for eight years; it then was part of the North American continent. Our membership brochure says that ISPUG dues in North America are $15 per year, and that overseas dues are $25. Canadians keep sending us $25. Sigh. Guess we better head North and check for a new ocean. If non-swimmers send us $25, we either remit $10 or extend membership for 20 months...

BACKUP POWER Our local power company we call The Mickey Mouse Electric Power and Candle Co.; not a week goes by but we are browned- or blacked-out. Seems that ocean salt deposits on the transformers; they short out if it doesn't rain, and burn down the poles. We've therefore had backup power for our computers and drives since 1980, and haven't lost a single byte to power problems in all that time. Our first rig was internal battery backup, a unit no longer made or supported. When these units began to show signs of old age, we knew we'd soon have to replace them, and decided to get an external unit, which should have the advantage of being useful (if properly chosen) with any future computer or disk drives. So we researched what was available, how it performed, what it cost, and report the results:

A backup unit (called Uninterrupted Power Supply, or UPS, in the grand tradition of saying janitors are sanitary engineers) should be sized to whatever rig you <u>plan</u> to have within three to five years; we thought we'd be wise to allow for a hard disk. If you do this, you soon find out that hard disks require much better performance from a UPS than do floppy drives.

Your first step is to determine the wattage output you need. SPET demands 117 volts at 1.0 amp; an 8050 drive, 117V at 0.5 amps. Well, 117V x 1.5 amps demands 175.5 watts. Printers and other heavy-demand devices shouldn't be put on a UPS. You find most firms include in their product line a small UPS with a 200-watt capacity, good for from 5 to 25 minutes at 200 watt draw (performance depends a lot on how often you use the disk drive, on how old the battery is and on how well charged by the charger inside the UPS). Then you find that the performance of the units varies all over the lot. Some (those made by Tripp Lite), are for blackout protection only, and let the computer crash at brownout (low voltage). We tried one and returned it. Brownout is more common than blackout (complete power loss).

Some units kick in when line voltage drops to 108 volts; some at 80. In our experience, the higher the voltage at which the unit kicks in, the better. A good unit should time delay return to the line (voltage often fluctuates during a brownout). We wrote six firms (Kalglo, Melrick, PTI, Qubie', SAFT, Sun), got the specs, tested the three units which seemed best in performance, and found that not a one, even the most expensive, would protect SuperPET against brownout. Half the time when the lights grow dim and the voltage flickers, SuperPET will crash, despite the UPS. Yet all units tested protected us against blackout with no problems. It's obvious that the units need a voltage drop cutout which simulates a blackout to be useful with SPET.

Concept: plug the cutout into the wall outlet; plug the UPS into the cutout. If voltage drops, the cutout switches off line power (simulates a blackout). We're darn sure it will work if we can find the cutout! So, help! If any of you know of such a cutout, please write or call. Tell us who makes it and where we can buy it, please. We'll pass the info along. Until we find a good cutout, we don't recommend that anybody buy a UPS for brownout protection on SuperPET.

CHAINING BUG IN BIG mBASIC PROGRAMS Frank Brewster long ago reported that mBASIC wouldn't pass matrices properly when programs were CHAINed. Waterloo went to work and issued a patch, which seemed to fix the problem (see I, 209). Now Frank reports that he wrote a huge program with a lot of string matrices; that some were passed when he CHAINed, and that some weren't. He found no predictable pattern in the failure to pass string arrays to the new, chained program. If any of you encounter this problem, please send us a disk copy of the programs so we can explore it and pass a defined bug along to Waterloo.

OS/9 PROBLEM WITH THREE BOARD SUPERPETS SOLVED We reported previously a mysterious problem in which OS/9 failed to run on some early, 3-board SPETs. Guru Avy Moise of TPUG and our own guru Terry Peterson collaborated to diagnose and cure the problem. It seems that the refresh of the 64K of banked RAM on these early 3-board machines is handled differently than the refresh of the 32K of user memory (refresh means that the transistors in RAM are periodically recharged to maintain their state; if this isn't done, electrons leak and memory goes phhhht). The refresh in the 64K switched banks depends upon the 60-per-second interrupts which, among other duties, sense the keyboard; if these are suspended long enough, bank RAM is not refreshed--and goes blaaah. The loading routine for OS/9 suspended such interrupts (SEI for you assembly language types) just long enough for this to happen. Once he and Avy and diagnosed this, Terry P. revised his loader to enable interrupts during loading and called us to happily report that the problem is gone. Congratulations to Terry and Avy for a fine piece of sleuthing. Anyone who has trouble loading OS9 on an old 3-boarder should get in touch with TPUG to get a copy of the new loader.

SHARP DEALER Bill Cronkhite of Pasadena tells us that when he ordered a new Amiga with 512K, he asked if the dealer stocked 3.5-inch double-sided disks. The dealer replied, "No. Why do we need those?" Why, indeed. Well, give the dealer credit; at least he knew Bill wanted to buy a computer, not a girl friend.

2031 ADDRESS CHANGES Last issue, we covered address changes on 4040, 8x50 and 1001 drives (p. 166 ff). Paul Matzke sent us address change data for the 2031, which we show at left. Data in < > are ASCII code numbers (see article last issue for using them in BEDIT). All commands except the first <119> are the

```
Change device from 8 to 9
M-W<119><0><2><41><73>
```

Change device from 9 to 8
M-W<119><0><2><40><72>

CR18	CR19	Device No.
in	in	8
out	in	9
in	out	10
out	out	11

same as those used on the 4040 and 8x50. Hard wired address changes are made near the middle of the 2031 circuit board at diodes CR18 and CR19. By removing them in the proper order, you can get several different device numbers, per the table shown at left.

The M-W commands are software commands; a 2031 drive will default to device 8 if you switch to 6502 or turn the drive off. Diode changes are permanent. As we wrote this note, we received a letter from Dr. J.G. Cordes, who has an MSD drive; he says that the M-W commands for changing device number are the same as those for the 2031, as shown above.

A GLANCE OVER THE FENCE Whilst P.J. Rovero has a sick SuperPET, he's been using some G.I. Zenith PCs running MS DOS, and comments: "I've been frustrating myself on MS DOS. The interpreter environment provided by Waterloo is so much nicer than the Microsoft Basic and Fortran compilers. Basic programs which run without error in the interpreter fail regularly after they're compiled, with no meaningful location of the problem. There's no Fortran interpreter, so all the debug routines have to be recompiled into the program under test. What a pain in the ---! To be sure, there are a few nice things: compiled programs (if they work) are very fast, and the Zenith Basic Screen Editor (BSE) is very powerful and easy to use (it's an editor for everything, not just Basic). I've used its macros to reformat data files output by one program as input for another."

GHOSTS BEYOND THE RIGHT MARGIN In the Waterloo MicroEditor, or in any form of Joe Bostic's BEDIT, write a full line of 80 x's. Then give a CHANGE command: c /%^/YYYYYYYYYYYYYYYYYYYY. The change will preface the first x, at the start of line, with with 20 Y's. Question: How many x's remain on the line? If you say 60, we're sorry to say you're wrong. You still have 80, but you can't see the last and rightmost 20. You can prove that with another CHANGE command: c*/Y/ --which will change all Y's on the line to nulls. Lo!--you again have 80 x's on the screen.

Now, try the same thing, using the DELETE key instead of the second CHANGE command. Sorry, Charley, you lose the rightmost 20 x's. Lesson: when using CHANGE commands, you can easily create hidden text and save it either to disk or to printer. There's an easy way to get rid of such hidden text: c80/%.%*/ , which says to change the 80th and subsequent characters on a line to a null (%. is any character, %* is the previous character, repeated 0 or more times). Is there any other way? Yes, you can edit a line with any key. Any change made will eliminate the 81st and subsequent characters. Lines may be up to 254 characters long...

If you think the ghost characters are a nothing but a problem, we suggest a few ways in which they're useful. Suppose you get over-80-character lines from a teleconversation and want to print only the middle of the file. Get it into the editor, delete the start and end; send the remainder to printer unedited. If your printer is set to do a CR when the carriage hits the right-hand carriage stop, you can print the long lines.

BEDIT makes the long lines even more useful. Suppose you want to underline some words in text (we won't complicate this with boldface), and further that you use the accent grave (`, or shifted @) to mark the start and end of underline, as we do on`underline these words.`Suppose further that your printer takes ESCAPE 4 as the command to start underline, and that ESCAPE 5 commands the end of underline. We now issue a CHANGE command: *c /`/ <27>4, which says to change the first of the accents to the "start underline" sequence. We follow that with another command to change the second accent to a STOP underline. Then we PRINT the page to printer. The underlined line above looks like this, on screen, at output:<

do on <27>4underline these words.<27>5 Suppose further that your printer takes E

Though the original text is pushed beyond the right margin, it's still there. We print the line despite the fact that we cannot see it all. Of course, you would not do this manually, but with an EXE file. We're sure some of you'll find even better ways to use the long lines capability in the editors.

APL CHARACTERS ON THE 4022 PRINTER Paul Rundle of Ajax, Ontario sent us a disk containing programs which will output APL characters to a 4022 printer. We always test programs before we issue them on ISPUG disks, but have no access to a 4022. Whoever asks first (and promises a test report) will get a copy of the disk for free. Write ye editor.

SAY GOODBYE TO ACCIDENTAL BYE
or,
How to Patch OFF

Roger Bassaber, our faithful correspondent in the distant Indian Ocean, sent a small program which changes the BYE in any language to OFF. If ever you lost a program in mBASIC by by saying BYE to the editor and instead kissing your work goodBYE, take the two minutes needed to run the program below. Forever after, you exit mBASIC only on an OFF. Roger, preferring OFF, has patched all his languages to quit on OFF.

We located two BYEs in mBASIC, one beginning at $0b9b; the second at $80f3 bytes from the start of the disk file. The first controls exit from language; the secund from the microEditor. The program below patches the language exit command to OFF; you can as easily change the mED exit command to <u>any</u> three-letter word you may fancy. Roger reports he has patched Cobol, Fortran, and Pascal. Revise the program below to stop searching for or patching BYE after it is first found in any of those languages.

Note that we get a long string of code at each gulp (using LINPUT), ending the string only when LINPUT encounters a CR. This way, we input several hundred to several thousand bytes at a time. The program runs in about two minutes. If you substitute GET (and input one character at a time) you'll need some 30 minutes.

```
100 ! fastoff:bd.
110 print chr$(12);
120 input "Enter language to patch (in upper case): ", lan$
130 open #2, "disk/1." + lan$ + ",prg", input     ! Old mBasic in drive 1.
140 open #3, lan$ + ",prg", output                ! New file created on drive 0.
150 print chr$(12);"Bytecount is:";
160
170 on eof ignore
180 loop
190   linput #2, line$
200   if io_status then quit
210   if bytecount < hex('0b9e')            ! Limits search only to language BYE.
220     x=idx(line$,"bye")
230     if x then line$(x:x+2)="off"  : print "Found!";
```

```
240    endif
250    bytecount=bytecount+len(line$)+1    ! Add CR, not counted in LENgth.
260    print bytecount;
250    print #3, line$                     ! Add CR stripped on input.
260 endloop
270 print #3,line$;                        ! Print last line without CR.
```

'Twould appear that mBASIC commands are in table form, and that you may amend the table by simple substitution. Neither BYE nor OFF can be abbreviated; both are three-letter commands. We'd suspect that anybody who tries to substitute ERAse for the word SCRatch, or to amend any command with another of different length, may end up in deep trouble.

THE ONLY VIRTUOUS VIRGIN IN THE LAND or, Perceptions of Shapely Structures

The fairytale goes that a stupid king, having found a virtuous virgin, proclaimed her the only one in his land, which so enraged the rest of the virtuous virgins that they conspired to have her burnt at the stake as a witch. We can't find any virtuous virgins any more (except for that extremely ugly third-grader down the street), but we do know how outraged they felt. It isn't fair to assign any particular quality solely to one person--or to one computer language, for that matter. Yet it doth appear that there is a conspiracy to assign all virtues to that aging virgin, Pascal; paramount amongst those virtues being her structure. Frankly, we don't think the old girl's structure is as attractive as that of some of the younger languages, which are shapely indeed. We'll demonstrate what we mean and then examine the idea that you simply must study Pascal to grasp other, structured languages, such as 'C'.

In Gonnet's book on algorithms and data structures (see separate article this issue) we found the following Pascal fragment, which we copied most carefully (semicolons and all) and later translated to microBASIC. It's part of an algorithm to perform an interpolation search. Is the structure explicit and clear?

```
while (set[high].k => key) and (key > set[low].k) do
      begin
      index:=trunc((key-set[low].k)/(set[high].k-set[low].k) * (high-low)+low;
      if key > set[index].k then low := index + 1
      else if key < set[index].k then high := index-1
                else low := index
      end;
if key = set[low].k then found( set[low] )
                else notfound ( key );
```

As with most structured languages, Pascal may be formatted to the user's whims. Will the WHILE loop be lucidly explicit no matter how the lines are formatted and indented? (see an ENDWHILE?); are the ends of IF equally resolute and clear? Dangling and ambiguous ELSE clauses are one of the curses of the language. The problem is easy to define: Pascal often does not explicitly end its structures; the end must be inferred. Often in published Pascal code we see the structure dive into an abyss of uncertainty, never to be seen again. This is a paragon of structured languages? Let us compare the structured microBASIC translation:

```
while (set(high) => key) and (key  > set(low))
   fraction = (key-set(low)) / (set(high) - set(low))
   index = ip(fraction * (high-low)) + low
   if key > set(index)
      high = index + 1
   elseif key < set(index)            ! Comment: Somehow this language
      high = index - 1                ! manages very well with nary a
   else                               ! semicolon to mark the end of any
      low = index                     ! statement. Why does Pascal demand
   endif                              ! them? So you can chase syntax errors?
endloop
if key = set(low)
   print "Entry found"
else
   print "Not Found"
endif
```

The beginning and end of each structure is explicit. No matter how the program is formatted it is not ambiguous. Pascal teaches good structure? What does the second example teach? Somehow, we find the second virtuous virgin a bit more virtuous than Pascal. Lately, we've been exploring some other languages: True Basic (written by the folks who wrote the first BASIC), BetterBasic (which owes much to Pascal), Basic 09 (very similar in form to mBASIC), and 'C' itself. Very frankly, we think the structure in most of them knocks the virtuous and structured socks off old dame Pascal. She never was very pretty; she is garrulous and an utter shrew about data types and syntax.

It appears we are not alone in this opinion. Glenn Hart reviewed True Basic in the October '85 issue of Creative Computing, and said: "True Basic includes structured programming constructs superior even to those of Pascal or C." If someone out there should object that no Basic ever can match Pascal's data types and matrix handling, please read the review of BetterBasic in the October Dr. Dobbs, which concludes that BetterBasic matches Pascal in both features--and has superior structure. In short, who needs grumpy old dame Pascal?

There are a lot of other virtuous, well-structured virgins roundabout. It is high time their shapely structures and accomodating natures were appreciated; that academics and computer-science types, saturated in Pascal, realize that time is passing by that fussy crone, even though she is dearly beloved by curriculum directors from coast to coast.

* * *

STRUCTURE AND 'C' We became interested in 'C' many months ago, and finally screwed up courage enough to read "The C Programming Language," by Kernighan and Ritchie. We expected it to be dull, complex and academic, written in High Abyssinian Greek; we were astounded to read simple, lucid text well-illustrated with copious examples--a model of its kind. So we bought "The C Programming Tutor," by Wortman and Sidebottom (not all Sidebottoms are bankers, Bodsworth) and again hit the jackpot. We've not read a simpler, clearer, better tutorial for any language. (If you're interested in 'C', we recommend both books highly.) At its simpler levels, 'C' turned out to be easy if you are familiar with any language which is structured.

We found 'C' easy to follow once you understand how 'C' uses curly braces to demarcate a block of code. We demonstrate with part of a binary search algorithm in microBASIC and in 'C' (the /* */ symbols mark a comment in 'C'):

In microBASIC

```
loop
  mid% = (low% + high%) / 2
  if search$ < tableword$(mid%)
    high% = mid% - 1
  elseif search$ > tableword$(mid%)
    low% = mid% + 1
  else
    found% = 1
  endif
until low% > high% or found%
```

In 'C'

```
while (low <= high) {
   mid = (low + high) / 2 ;
   if search < tableword(mid)
      high = mid - 1 ;
   else if (search > tableword(mid))
      low = mid + 1 ;
   else
      return (mid) ;  /* see below */
} /*this curly brace means 'endwhile'*/
return (-1) ;          /* see below */
```

Once you are accustomed to the notion that curly braces set off the beginning and end of a block of code more than one line long, the similarity in structure of the two languages is striking. The commented RETURNs in 'C', above, are instructions to send the values stated back to the calling routine and to QUIT the function. If search succeeds, RETURN sends back the value of the variable "mid". If search fails, it sends back a -1. We don't see why anybody must study Pascal to understand 'C', especially when 'C' is not strongly data typed and Pascal is.

If you get to know any of the virtuous structured virgins, you should be able to snuggle right up with 'C'--except, perhaps, for the virus of the semicolon pox, which evolved on the right margin of an ancient teletypewriter or perhaps on a 40-column screen...and then festered into the present plague.

* * *

THE EVOLUTION OF THE SEMICOLON POX Breathes there a programmer who has ever written a two-page program in 'C' or Pascal without a syntax error generated by a missing semicolon? Confine yourself to 40 columns or less and say:

```
  writeln('This is a long print statemen   {Woops! We're at right margin!}
        t.');                              {So we use another physical line}
```

Early on, the decision was made to end logical lines with a semicolon so that long statements could occupy two or more physical lines. The Pascal program at left uses one "writeln" statement to print four physical lines to screen, and does it within the limits of a 40-character terminal. Obviously there was another option (sadly, it wasn't taken) to mark the continuation of a logical line by a special symbol, such as the ampersand "&". Some languages take this opposite route; a continued line then is marked like this: writeln('We con'& &'tinue this line')--which needs no semicolon. Now that the 80-column screen is with us, do we really need a semicolon to end every complete statement?

```
begin
  page;
  cr:=chr(13);
  writeln('Physical Line 1,',cr,
  'Physical line 2,',cr,
  'Physical line 3,',cr,
  'And a 4th, ended by a semicolon.');
   ...
end.
```

Because we're no longer squeezed for space, it'd be far easier to use an occasional & or \ on those rare long lines, rather than drudging in a semicolon on every complete statement. Sigh. And so doth the past inflict upon us the semicolon pox, from which God deliver us one day. We doubt that the pox ever will be cured in Pascal, for, having chosen semis, Niklaus Wirth then connected such structures as IF...ELSEIF...ELSEIF...ELSE as a single logical statement, the end of which you must define with a semicolon. Life need not be poxed--a number of languages handle the same constructions with nary a semicolon, which indeed does prove that the semi is a pest born of poor design.

Unfortunately, old habits die hard. It may be decades before the semicolon pox is extinct. But there is hope. Ever notice that you can TAB only from left to right on your SuperPET? Why? It's a habit carried over from mechanical typewriters, in which a coil spring pulled the carriage from left to right. There was utterly no way to TAB from right to left, against spring tension. Thirty years after the mainframe computer was introduced, someone finally realized that computers don't use carriage springs... A few computers now let us use SHIFT TAB to tab from right to left... The past has a long, cold grasp, not easily dissolved by time.

A PROGRAM TO SEND FILES TO A PARALLEL PRINTER

by P.J. Rovero

Last issue, we printed a program received from Josh Rovero showing how to use SuperPET's user port for output to devices employing the Centronics parallel protocol; we print this issue a program which you may assemble and link to output any disk file to a printer employing that protocol. It loads and runs at main menu. See last issue for definions of registers, procedures, and wiring diagrams.

The user port connectors you'll need are available from AB Computers [252 Bethlehem Pike, Colmar PA 18915 : (800) 822 1211]. The price in the latest catalog is $2.30. Centronics plugs and ribbon cables (40 conductor) are available pretty widely from electronics suppliers. The 6522 VIA chip is also available from AB Computers for $5.00. Josh hasn't had to replace it in the 8032/SPET, but did replace it on his Pet 2001. So long as only TTL voltages (0-5) and loads are seen by the VIA, he advises you should not have any problems. Note that Josh employs John Toebes' CALL MACRO, as published in Vol. I, page 158.

```
;Program sends file thru simulated Centronics port. By P. J. Rovero, 20 May 85.

;Accomodates either NSTROBE/NACK or NSTROBE/BUSY protocols. Connect the NACK or
;busy line to CA1 (user port pin B). Consult your printer manual for guidance on
;choice of protocol.  Gemini 10-x works fine with either one noted above.

xref fputchar_,putchar_,fgetchar_, getrec_, printf_, putnl_,openf_,closef_
xref errorf_,eof_

zero        equ  $0000          ;Constant.
service_    equ  $0032          ;Return to menu w/o reset.

                     ;VIA addresses
port_Aca1   equ  $e841          ;Port A with CA1 handshake.
ddra        equ  $e843          ;Data Direction register.
acr         equ  $e84b          ;Auxiliary control register.
perif       equ  $e84c          ;Peripheral control register.
ifr         equ  $e84d          ;Interrupt flag register.
port_A      equ  $e84f          ;Port A without handshake.

;include<call_macro>
```

```
start       call printf_,#title                ;Print title; ask for input.
prompt1     call printf_,#quest1
            call getrec_,#in_file,#16
            std  len_in
            addd #in_file
            tfr d,x
            clr ,x
            jsr putnl_
            call openf_,#in_file,#in_mode      ;Open the file
            bne  success1                      ;successfully, or report
            call printf_,#error1               ;an error.
            jmp  prompt1
success1    std  in_byte
            jsr set_via                        ;Set up for output.
            ldd newline
            std char
            jsr send_it
            loop
                 call fgetchar_,in_byte        ;Print the file.
                 std  char
                 call eof_,in_byte
                 quif ne
                 jsr send_it
            endloop
wrapup      ldd newline                        ;Clean up and close the file.
            std char
            jsr send_it
            call closef_,in_byte
            call printf_,#final
            lda t_acr             ; Restore VIA auxiliary control and peripheral
            sta acr               ; registers.
            lda t_perif
            sta perif
            ldd  #zero
            std  service_
            rts

set_via lda #255
        sta ddra                       ;Make all port_A lines outputs.
        lda acr
        sta t_acr                      ;Save contents for restoration.
        anda #%11100011
        sta acr                        ;Disable shift register.
        lda perif
        sta t_perif                    ;Save contents for restoration.
        anda #%11111110                ;Set CB1 for neg transition.
        sta perif
        ora #%11000000                 ;Set cb2 for manual control.
        sta perif
        ora #%00100000
        sta perif                      ;Put CB2 high.
        lda port_Aca1                  ;Clear the flags.
        rts
```

```
send_it ldb char+1
        stb port_A                    ;Put data on output port.
        lda perif
        anda #%11011111               ;NOT 32.
        sta perif                     ;Pull cb2 low for a couple
        jsr delay                     ;of microseconds.
        lda perif
        ora #%00100000
        sta perif                     ;Put CB2 high.
test5   lda ifr
        anda #%00000010               ;Has nack/busy had negative transition?
        beq test5
        lda port_Aca1
        rts

delay   pshs x
        ldx #$0002                    ; Delay of about (19+(8*x)) microseconds
        loop
           leax -1,x
        until eq
        puls x
        rts

t_perif    rmb 1         ;All data needed by program
t_acr      rmb 1
in_file    rmb 20
           fcb 0
in_byte    rmb 2
out_byte   rmb 2
char       rmb 2
len_in     rmb 2
len_out    rmb 2
in_mode    fcc "R"
           fcb 0
title      fcb 12                               ;Clear screen.
           fcc "Disk file to Centronics thru VIA"
           fcb 13
           fcc "by P. J. Rovero     20 May 85"
           fcb 13,13,0
quest1     fcc "Input filename ?"
           fcb 13,0
error1     fcc "Error opening input file.  Try again."
           fcb 13,0
newline    fdb $000d
final      fcc "File has been sent thru VIA."
           fcb 13,0
           end
```

HIEROGLYPHS AND ICONS or, The Consequences of Keyboard Machismo Part I

The interface between people and computers has for years seemed to us both clumsy and badly conceived. Our first computer ran CP/M, which was a popular operating system a few years back; we couldn't wait to give it away (think about having to warm boot a computer every time you change a disk. That's CP/M!). After five years of asking why the designers of CP/M could blunder so badly, we suspect we've found a significant part of the answer: nobody who designs computers or operating systems comprehends how professional users (other than programmers) employ keyboards and computers.

For a long time, the only keyboard professionals were secretaries and authors. Either could have told Digital Research (who designed CP/M) that when you process words you must have instant access to every disk you own. Who among the DRI programmers (undoubtedly hunt-and-peckers) ever had to reorganize an entire book on a tight deadline; assemble an appeal from a jungle of source files--or leap from letters to invoices to dictation, as secretaries must, every hectic day? If any had, CP/M would have been far different. CP/M was designed by hunt-and-peck programmers for hunt-and-peck programmers. All computers (and most computer keyboards) are still designed by hunt-and-peckers for hunt-and-peck use.

What has the hunt-and-peck viewpoint done to keyboards? Gaze upon the keyboard on SPET. Put your right-hand fingers on the 7-8-9-0 keys, where they're supposed to be when you touch-type numbers. Is the RETURN key L-shaped and big enough so your little finger can press it that position? Nah. Yet you can do so on that superb old workhorse, the IBM Selectric, which has the best keyboard ever made. Why is the + sign substituted for the colon key? Why are the brackets [] and curly braces {} (and a lot of other keys) in non-standard locations? The designers didn't know any better.

SPET's keyboard is better than that on most computers. Yet, with the exception of a few custom keyboards made for the IBM PC (and, praise be, the Selectric layouts on the new Amiga and the Atari ST), there isn't a decent keyboard on any computer we've ever seen.

This issue, we explore the effect of the keyboard as an input device on individual users. Next issue, we'll look at the problem from a broader aspect.

* * *

The Crippled Professional What we now say applies to those who use a computer professionally--programmers, authors, editors, analysts--anyone who sits at a terminal most of the day. Suppose you are such a professional. Suppose further that your legs are perfect but you've never learned to walk. Would you crawl about the office on your hand and knees for thirty years--on the excuse that you couldn't take the time to learn how? Hardly!

Yet we know an attorney who works long hours each day, hunting-and-pecking at his PC; he'll probably be two-fingering the keyboard twenty years from now. Not far down the road are two professional computer programmers who still hunt-and-peck all they write. Some professional women hunt-and-peck (H&P), refusing to learn touch typing, because they refuse to be classified as "secretaries."

We're told by men that "executives don't type." In the same sense, Grand Dukes in 1890 would not condescend to drive their own carriages, although their grandsons are fairly supple at the wheel of a Fiat. Typing has for so long been a female, menial, and low-status task that men are repelled by a keyboard. That will change, in time, as the computer becomes universal and as skill in using one becomes another focus of the eternal competition between individuals.

How much time does a H&P typist waste? We tried to find out, but must generalize. It depends on whether you use keyboard macros, on whether you enter text or program. If you program, it depends on language. 'C' is most concise; COBOL is not. With the cooperation of the pros, we ran some trials and generally concluded that touch typists program from twice to four times as fast as their H&P brethren. Writers who touch type are from four to ten times as fast as authors who hunt-and-peck.

Let's assume that only half your workday is spent typing, even if you will sit in front of a keyboard most of the day for the next 20 years. How many years will you waste if you H&P? Well, to be conservative, a programmer throws five years of his life down the tubes. A professional writer, with equal conservatism, wastes at least eight years--and probably more. In a competitive world, who can afford this terrible waste of time?

We suspect most H&P typists never think about it. They are in a sense crippled, but not as obviously as someone crawling about the office on hands and knees. Why don't people learn to touch-type, just as they learn to drive or cook? It's a new problem, brought on by the computer. What was once a feminine, low-status, secretarial art has become an essential skill. People haven't faced up to it. And the macho, executive image is still a strong deterrent. That too will pass, as did the Grand Dukes, as did their liveried carriage-drivers.

It seems obvious to us that touch-typing has become as essential a skill as reading or writing--if you expect to compete. Iconographic operating systems, such as those on Macintosh, don't bypass the issue. Sooner or later, you must enter text from a keyboard, icons notwithstanding.

Ah, well; some of us may think that computers will soon be equipped with intelligent speech recognition systems. We can see ourselves struggling with speech recognition on a new computer:

Us: "Change 'missed her' at page bottom to 'Mister', that's capital M, small r."

Computer: "Where it says 'missed her brown'?"

Us: "Yes, and change 'brown' to 'Broun', capital B."

Computer: "Mr. B.?"

Us: "No, dammit; capital B lower case r-o-u-n!"

Computer: "Okay. Mr. Broun."

Us: "Change o-i-l in line 1 to 'Earl'. That's capital E small a-r-l, okay?"

Computer: "Okay. Change 'oil of Norfolk' to 'Earl of Norfolk'."

Us: "Change the 'do be us' on line 12 to 'dubious'."

Computer: "Okay. Change 'do be us Dutchess' to 'dubious Dutchess.'"

Us: "Change 'breeches' to 'breaches'."

Computer: "Where it says '...fought his way into the breeches...'?"

Us: "Yeah. He was attacking a city, not attempting rape. Now change to 's-e-e,' not the ocean 'sea'! And change that 'too' to number 2."

Computer: "Which 's-e-a,' which 'too'?"

Us: "Lessee, line...14. That 's-e-a.' Line...23, that 'too'."

Computer: "That also is a s-e-a? I fail to find a s-e-a on line 23."

Us: "@##*#! @##**@*@!" [As we turn off the speech recognition software...]

We expect the keyboard will be with us for quite a few years. Should speech recognition come to market, how long will any of us remain sane when all about us people curse incessantly at their computers?

* * *

How Do You Learn Touch Typing? Our handwriting is the worst in the Western World. Because neither our teachers nor our parents could read it, we were dragooned into a typing class, the only boy among 46 girls--a matter of small embarrassment and great opportunity. It was all we could do to keep our hands on that old black Underwood and off the girls... We were touch typing--if slowly--after some 30 hours of practice; our little black book was full. In retrospect, those 30 hours were the best investment (social or professional) we ever made.

We hestitated to run this article without offering a solution for SPETters who want to learn to type, and so wrote a program called SuperGRIT which will teach you to type (if you have the grit to stick with it). Then we hesitated again, fearing that readers would think we were trying to peddle disks. We've been debating what to do for several months. Then we read articles in ComputerWorld and Personal Computing which pulled no punches on the necessity for middle managers to learn how to use the keyboard. That did it. We decided to run the darn article and offer the disk, which provides 25 lessons. If you have the grit and if you persist, you should be touch typing in 30 hours of practice. You won't be fast, for speed comes with time, but you will type faster than you formerly hunted-and-pecked.

SuperGRIT has three lessons on touch-typing the keypad, which is simple and easy to learn. The program remembers what lessons you practiced, and for how long. You are drilled on all keys you miss, and receive a report on how well you've done at the end of each lesson. We included a timer to tell you to stop and rest, because conditioned reflexes don't form when you're fatigued from too much practice. Write Editor, PO Box 411, Hatteras, N.C. 27943 for SuperGRIT, 4040 or 8050 format. Price $10 U.S. Money back if you don't like it. All proceeds go to ISPUG, not to the author. We promise you will loathe and abominate us when about halfway through. To persist, you'll need grit.

BINARY CONVERSION ROUTINES FROM THE OLD MAESTRO The best way we know to get neat, tight code on any problem is to publish a long routine which will drive all good assembly language programmers up the wall. You then receive small, simple masterpieces which are yours forever after. We got a bunch on the binary conversion routines published in II, No. 5 (p. 126), and print the best of them below. Those we picked as the shortest and best came from the old maestro, Joe Bostic, who wrote BEDIT. As you scan them, you'll see why BEDIT is so fast--Joe's code is short, sweet and simple; it leaves you wondering why you ever did it the hard way.

First, here's his version of a routine to convert an eight-character binary string to a counting number which is returned in two bytes (high byte clear). In this example, as in all that follow, he assumes that the binary string was previously checked for entry errors, and that a buffer named "buffp" is defined:

```
bin2cn  LDX  #buffp    ; Index the start of the binary string; could be
        CLRB           ; TFR d,x if parm is passed to routine in D register.
        LOOP
          LDA  ,x+
          QUIF EQ      ; Stop on endstring null. Note that $30 (ASCII 0) has
          LSRA         ; a 0 in the rightmost bit; $31 (ASCII 1) has a 1. So,
          ROLB         ; shift the rightmost bit to the Carry Flag; then ro-
        ENDLOOP        ; tate the carry flag into B register, which holds the
        CLRA           ; counting number.
        STD buffp      ; Store counting number in same buffer.
        RTS
```

The next routine converts a binary string of from eight to 16 characters into a two-byte counting number. It leaves X and Y registers unchanged, and will stop even if you forget to put a null at the end of the binary string. He assumes a call with D register holding the address of the string to be converted.

```
bin2cn  PSHS  x               ; Preserve whatever is in X.
        TFR   d,x             ; Set up index register to string.
        LDD   #0              ; We'll put the result in two bytes on stack,
        PSHS  d               ; which we clear.
        LOOP
          LDA    ,x+          ; Get a character,
          CMPA   #'0
          QUIF LO             ; quit if it's less than $30 (ASCII 0), or
          CMPA   #'1
          QUIF HI             ; if higher than $31 (ASCII 1).
          LSRA                ; Shift first bit to carry flag, as in first
          ROL    1,s          ; routine. Then rotate carry bit into the
          ROL    ,s           ; two stack bytes, which hold the number.
        ENDLOOP
        LDD   ,s++            ; Return with the result in D register.
        PULS x,pc             ; Retrieve X, and get address of next instruction
                              ; into the Program Counter (in effect, RTS).
```

The next routine reverses the process, and converts a counting number back to a binary string. It normally outputs eight binary characters, but if the number should be larger than 255 it outputs 16. When you call the routine, put the number to be converted in the D register. Joe's original passed the buffer address as a parameter; we took it out because it obscured the conversion process itself. Y register is left unchanged; X might easily be stacked if it too must be preserved.

```
cn2bin  PSHS d        ; Stack value to be converted (passed in D Register)
        TSTA
        IF NE         ; Do we have 8 characters or more than that?
          LDB #16     ; Yes, do full two-byte answer.
        ELSE
          STB  ,s     ; Stuff low byte into empty high byte, on stack.
```

```
    LDB  #8      ; Do 8 bits only.
  ENDIF
  PSHS b         ; Use top of stack for a decrement register.
  LDA #'0        ; Set up ASCII zero and one,
  LDB #'1        ; so we can make copies.
  LDX #buffp     ; Point to buffer address.
  LOOP
    LSL  2,s     ; Roll a bit out of the source value high and low
    ROL  1,s     ; bytes and convert. See example below for what happens.
    IF CS        ; If the Carry Flag is Set,
      STB ,x+    ; it's a 1, so store ASCII 1 in buffer.
    ELSE
      STA ,x+    ; or buffer an ASCII 0.
    ENDIF
    DEC ,s       ; Count down to zero
  UNTIL EQ
  CLR ,x         ; End the string in a null
  LEAS 3,s
  RTS
```

It isn't obvious how the LSL and ROL instructions work in the programs above, so we diagram a few cycles below. In effect, the high byte is read first; the low byte values are shifted to the high byte and read last. Start reading at the right margin on each line; move to your left for each step on that line. A LSL (Logical Shift Left) feeds a 0 to the rightmost bit in a location and puts its most significant bit (the high bit) in the Carry Flag. A Rotate Left (ROL) feeds the intermediate Carry Flag into the low bit of the high byte and <u>then</u> passes its own high bit to the Carry Flag. We use the hex value 00FF (binary 0000 1111) to illustrate what happens as the two stack values exchange bits through the Carry Flag. Each line shows the result <u>after</u> execution:

```
        String                          ROL                            LSL
Pass    Result        Final Carry    High Byte     Intermediate      Low Byte
No.     Stored:          Flag         (at 1,s)      Carry Flag       (at 2,s)

At Start                             0000 0000                       1111 1111
1         0     <--      0     <--   0000 0001    <--     1    <--   1111 1110
2         0     <--      0     <--   0000 0011    <--     1    <--   1111 1100
...
8         0     <--      0     <--   1111 1111    <--     0    <--   0000 0000
         (The eighth values were just shifted to the carry flags)
9         1     <--      1     <--   1111 1110    <--  all shifts are now zero.
...
16        1     <--      1     <--   0000 0000
```

This is bit-twiddling indeed; once understood it is a handy trick. You certainly do not write such code intuitively!

TABLE LOOKUP, SEARCHES, HASHING, ORGANIZING DATA and OTHER FUN STUFF
Part I

Collecting data is simple. Organizing it so you can find one item amongst the mass is a headache. If you try to organize data on your money, your students, inventory, stars, snails, postage stamps or the reigns of kings, you somehow have to order the data so you can quickly find what you want. For some unaccountable reason, there are few references in plain English on how to organize and search for information. The references we have found are written in High Abyssinian Greek (HAG) by academics; algorithms are buried in pages of higher math, in Pascal, "C", or psuedo-code which is never illustrated by example (academic types lose guru status if they write clearly).

If you have the patience to decipher HAG, you find a wealth of simple and powerful ways to organize and search data. The subject is intensely practical, no matter what kind of information you must manipulate. Our work to date has repaid us tenfold with new, swift and accurate ways to store and find data, both numeric and string.

We know where this series starts, but haven't the foggiest notion where it will end, since each new exploration opens new doors. We welcome algorithms, comments and articles from readers as the series stumbles onward.

* * *

ORDER AND SEARCH It is fundamental that you cannot decide how to organize information until you know how you will search it. The order and the search method are wholly interdependent.

There are two fundamental ways to organize data: 1) as it randomly arrives, in the order it arrives, and 2) in more ordered form (numeric, alphabetical, or hashed). We are continually surprised by the effectiveness of sequential order (storage in order of arrival) and of sequential search (start at the top of a list, search to the end) when the user is clever about it and lists don't get too large--particularly in slow, interpreted languages where complex code runs slowly and simple code is relatively fast.

For large lists, you must order the data alphabetically, numerically, or by a hashing algorithm (more later) if you must find it quickly. There is no such thing as an optimum method to order or search. The method you employ depends upon language, the data, and how you itend to manipulate it. Before you commit yourself to any method, consider and test the alternatives!

Anybody who wants to suffer horribly can read "Handbook of Algorithms and Data Structures," by G.H. Gonnet, Addison-Wesley, 1984, ISBN 0-201-14218-X. Don't say we didn't warn you. We also plucked (read "stole") some ideas from a library of magazine articles and from many a program sent by ISPUGgers. We'll illustrate each method with well-structured code and with examples, and will outline the weaknesses and strengths of each approach as this series goes on.

This issue, we cover two search methods: binary search and interpolation search. In both, data must be either in alphabetic or numeric order.

BINARY SEARCH
Simple and Fast

Binary search is a very efficient and simple way to locate an entry in an ordered table of numeric or string entries. By "ordered" we mean that all entries must be in numeric or alphabetical order in whatever data structure holds them (a table, matrix, relative file, etc.). The small table at left, below, shows an ordered table of acceptable numeric values. It could easily be a list of one hundred or one thousand selected entries, alphabetic or numeric. Our problem is to compare any outside value with the values in the table and to react properly if the value is or is not found.

Why is the method called binary? Because it continues to divide a table into halves, quarters, sixteenths, thirty-seconds...until a search key is found or isn't. The divisions obviously are the powers of 2. Let us search the small table at left for the value of 10. We first divide ten (the number of items) by 2, obtain five, and look at the fifth item, 40. Our key value (10) is less than that, so we know that it must must be in the lower half of the table (if there at all). We discard the top half of the table and have five entries left. We integer divide 5 by 2, obtain 2, and look at the second value, 22. Comparison again says that is too high. The second entry now becomes the top of the table; divide that by 2, obtain 1, compare--and we have found the value.

10 22 24 26 40 43 44 50 55 60

We show below a well-tested algorithm which will perform binary search. The key variables are defined below. We store the table in the elements of a matrix, and employ option base 1 (the matrix begins with element 1, not 0):

```
 loop
 ...
 high% = table_entries% : low%=1
 loop
   mid% = (low%+high%)/2
   if search$ < tableword$(mid%)
     high% = mid%-1
   elseif search$ > tableword$(mid%)
     low% = mid%+1
   else
     found% = 1
   endif
 until low% > high% or found%
 if found%
   found% = 0 : do something
 else
   do something else
 endif
  ...
endloop
```

high% = number of items in the lookup table.
low% = the number of the lowest element in the array; here, it starts as 1).

search$ = the value we attempt to locate.
tableword$ = any table value.

mid% = An integer which marks the element we examine in the matrix.

Binary search is efficient, the number of passes through a loop to find any entry increasing much less rapidly than the number of items in the table itself. The number of passes needed, where n = number of items in the table, is about

log(base 2) n

To make this concrete and simple, you may easily determine the maximum number of passes to find (or not find) any item in a table by using a table of the powers of 2, as shown below. Most searches will require fewer passes than the maximum:

Items in Table: (Powers of 2)	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
Maximum Passes:	1	2	3	4	5	6	7	8	9	10	11	12	13

Note that a search of a table of 63 items may be performed in six (not seven) passes; that of a table of 511 items in nine, not ten, passes. Though binary search is quite efficient, it can be beaten under certain conditions.

We found in John Toebes' new assembler a good example of binary table search, in which he compares possible keyword entries against a table of keywords. We show in "table," below, one table entry as an example of the format of all 250, which are in alphabetical order. At the end of the routine, to make it simpler to follow that routine, we show the stack arrangement. Note that the number of table entries cannot exceed 255. First, we show examples of the table entries and their labels:

```
table    ...
         FDB  ADMIT_,$013B,doadmit  ; 6 bytes: address of ADMIT_ in first two.
         ...                        ; Table holds a total of 250 entries.

entries equ (*-table)/6    ; Note the equate and the way the number of table
                           ; entries is found here and at **, below.
         ...
ADMIT_   FCS "ADMIT"       ; The label entry for ADMIT_, one of 250, to match
         ...               ; the table entries above. ADMIT_ is an address.
```

Next is the assembly routine to perform a binary search of TABLE. The addresses in the first two bytes of each TABLE entry point to the strings themselves. On assembly and linking, the label ADMIT_ defines the address of string "ADMIT".

```
;look.asm, by John Toebes, VIII.

reserved pshs  d           ;Stack address of string to be checked against table.
         clra              ;Set zero as lower limit of search.
         ldb   #entries-1  ;**Set upper limit (adjust for zero start of table).
         pshs  d           ;Stack upper and lower limits.
         ...               ;Do what is needed to process entries if found.
lookloop loop
            ldb  ,s        ;Get lower limit in B.
            cmpb 1,s       ;Compare lower limit to upper limit.
            LBHI nomatch   ;Branch to end when lower limit is the larger.
            clra
            addb 1,s       ;Add upper and lower limits
            adca #0        ;Compensate for results over 255, if any.
            lsra           ;Divide result by two,
            rorb           ;using both registers.
            pshs b         ;Save B as current index to table.
            lda  #6        ;Multiply current index
            mul            ;by 6 (bytes per item).
            tfr  d,x       ;Generate in X register an offset
            ldx  table,x   ;of 6*index for address of table string value.
            ldy  3,s       ;Get from stack address of search string.
            loop
               ldb  ,x+    ;Load table character.
               cmpb ,y     ;Compare with search string character.
               QUIF NE     ;No match; stop.
               tst  ,y+    ;End of string?
            until eq       ;Yes, endstring null. Characters check okay to end.
            puls  b        ;Recover the current index into table.
            quif  eq       ;If match, stop looking (backref to UNTIL EQ, above).
            if    hi       ;No match. Table entry larger. (Backref to QUIF NE)
               tstb        ;Is table index zero? We're at start of table,
               beq nomatch ;so quit.
               decb        ;No, make this location minus one new upper limit.
               stb  1,s    ;Current table entry is too big. Stack new limit.
            else           ;(Next line backref to QUIF NE; defaults to IF LO)
               incb        ;No, table entry too small. Increment current index,
               stb  ,s     ;make it lower limit, and stack it.
            endif
```

```
        endloop
        ...                 ;We have a match, so process it.
        ...
nomatch ...                 ;We have no match; do what is needed...
        ...
        rts
```

```
Stack Pointer Data:         Bytes Below Start        Value on Stack:
                               of Stack:
                     3,s       -01 and -02        Address of String to be Compared
                     1,s       -03                High Limit of Search
Usual SP-->           ,s       -04                Low Limit of Search
PSHS/PULS B Stack Pointer--> -05                  Revised Index to Table
```

SEARCH BY INTERPOLATION
Speed Demon for Number Crunchers

We wish we could broaden the range of this rocket, for it will find any number in a series of evenly distributed values faster than any other method (for gurus, the behavior is log log n). It requires even distribution of values because it calculates a decimal fraction of the number of items in the table (based on the search key), and then offsets to that position as the most probable location. Anybody who has "interpolated" a table of logarithms knows how it is done. If the algorithm doesn't find the value at the place calculated, it recalculates another position, much as binary search does, and looks again. Make no mistake, this method will find values in a table even if they are unevenly distributed, but it slows down.

Who would ever want to look up uniformly distributed data? Well, our examples, for simplicity, use integers. Suppose you have a table of decimal fractions; suppose you have calibrated an instrument and the values read must be changed to the calibrated values (e.g., 10 is really 9.8466, 11 is really 11.004, etc; you read 10 from the instrument, obtain 9.8466 from the second dimension of the table, and record the calibrated value).

This algorithm whips binary search hands-down if given appropriate data. You can get an idea of how uneven values slow it down by trying the two sets of data below; one is uniform (we found all search values in one pass); the second set is not (many passes were sometimes needed). In big tables, of course, non-uniform values slow this approach to a crawl, though you may sometimes evade the problem (see below).

For comparison, here are typical times to find a value or report it not found (in seconds) for tables of 40 values in mBASIC. In assembly language, this algorithm is superswift when properly used.

Distribution of Values:	Uniform	Gaps range from 1 to 6
Report Not Found (Value out of Table Range)	< 0.016	< 0.016
Report Found	0.018	0.018 to 0.284 (factor of 16 difference)
Report Not Found (Value in Table Range)	0.018	0.018 to 0.284

Beware the data types! Both "key" and "fraction" must be <u>real</u> variables. They create a decimal fraction of the table range at which the value wanted will be found or not found (hence the "interpolation"). If you must tabulate decimal values, you'll have to convert many variables in the program below from integers to reals; index% (the index to position in the matrix), low% and high% (indices to top and bottom of the table) must always be integers.

Sequential-Interpolation Search. Option Base 1.

```
  data 10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48   Uniform
  data 50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88   Values

  data 10,14,15,17,20,21,22,23,24,26,28,32,34,37,39,40,41,42,43,44   Not
  data 46,48,49,52,53,56,57,60,62,63,64,65,66,67,69,72,77,80,84,90   Uniform

dim set%(40)             ! set% is the matrix holding the data values above.
mat read set%            ! set may be REAL if data are in decimal form.
loop
  input "Enter search number: ", key            ! KEY must be a real variable.
  low% = 1 : high% = 40
  if key = ... then quit                        ! Choose your own QUIT value.
  while set%(high%) >= key and key > set%(low%)
    fraction = (key-set%(low%))/(set%(high%)-set%(low%))
    index% = (fraction * (high%-low%)) + low%
    if key > set%(index%)
       low% = index% + 1              ! The final search value, index%, must in
    elseif key < set%(index%)         ! turn be an integer. Well, yes, you can
       high% = index% -1              ! use the integer part of "index"...
    else                              ! In Pascal, "trunc" does the job.
       low% = index%
   endif
  endloop
  if key = set%(low%)
    print "Entry found"
  else
    print "Not found"
  endif
endloop
```

In our tests, we found a way to continue using this algorithm when we didn't have uniformly-distributed values. Whenever we encountered a value gap, we stuffed in a dummy, and in the second dimension of the matrix left a null value.

Assume matrix set% has two dimensions, the first one being the data value above, the second being the converted or calibrated values we want to use. To obtain uniform distribution, any "dummy" data value is followed by a null in the second dimension, which is then ignored. We retain log log n behavior at slight cost-- if the series of values doesn't have so many gaps we'd go mad trying to fill all of them with dummies.

READING SUPERSCRIPT FILES in the MICROEDITOR

by Martin Goebel

I have always thought it easier to prepare text in a word-processor rather than in an EDITOR, the reason being that a word-processor has line wrap-around, a feature not found in Waterloo's EDIT. [Ed. But found indeed in the various forms of ISPUG's BEDIT.] Quite by accident, I found a way to save text in a file readable by EDIT. The Superscript manual describes how to save files for BASIC, but of course they are gibberish in 6809.

The trick is this: rather than simply save to a file, save the Superscript text to a background print file (a disk file) with the original menu set to an ASCII printer. The output to the disk file is pure ASCII, which the microEDITOR can read. The detailed procedure is as follows:

1. Prepare your text in Superscript in the normal manner.

2. If you are using some other printer (as selected from the menu at the start of the session) reset it to the MX-80. This is done by pressing RVS and then CLR. Superscript will ask "Restart ?" to which the answer is "y". The main menu appears and the MX-80 printer should be selected. You will be returned to your text.

3. Now output the text by pressing RVS+O+C+S. The C is necessary because otherwise there is a problem at page breaks. Superscript will ask for an output drive # and then a filename. As with all 6502 files for later use in 6809, you should use only lower case letters in that file name.

4. That's it. The file can now be read into the EDITOR. Note that the file name will be in capital letters. I just rename the file according to ISPUG convention.

[Ed. The process of sending to disk what you'd normally output to a pure ASCII printer, as outlined above, is followed also for WordPro and Paperclip. For details, see Vol. I, 8, p. 95 and I, 14, p. 244.]

HOW TO ABOLISH A BOOBY TRAP or, NO MORE ACCIDENTAL RESETS

by Martin Goebel
79 Highland Drive, Wedgewood Park
Newfoundland A1A 3C3

Well, I finally got the nerve to do something that for years has frustrated me. I'm talking about that little bank of switches on the right side of SuperPET that flick it in and out of 6809 and 6502 modes. Commodore could not have located them in a more dangerous place. You dare not use the desk space on the right side of the computer. Pick up a pencil or reach for a manual and you almost inevitably flick a switch, RESETting to the wrong microprocessor and perhaps destroying hours of work.

It seemed to me that the switches belong in the space between the main housing and the screen monitor. There they are within easy reach of eyes and fingers but well-protected from stray pencils, rulers, books and desk debris which might send an evening's worth of work to data heaven. For those willing to risk all, here is how to put those switches in their rightful place. Two other hardware changes/additions described below are worth doing at the same time.

First, unplug the computer. Raise the top of the computer and lean it against a wall to provide ample working room. My machine is a three-boarder; after carefully noting the way the boards were connected, I removed them. The switches themselves were next; I recorded which one was which and the direction they toggled for their various modes. Drilling the correct size of hole through the computer cover from the inside is easy.

I selected a position along the right side of the ledge between screen and lower housing, about 1/2 inch in so that the switches would be tucked under the overhang of the monitor. I also set them back a bit from the edge of the ledge so that the original metal foil labels, when re-attached, would line up correctly.

The labels can be peeled off the switch bracket without tearing if you do it slowly. Now that I had the holes drilled I found that the wires on the switches were too short to reach from the boards. This was the real problem that had deterred me for so long.

The 6809 and the R/W switches plug into a circuit board. Unplug them; you may with the help of a pin pull each wire (with a metal connector attached) out of the connecting plug. I removed the wires from the connectors as well as from the switches and replaced them with longer wires. I also have switches for the UD11 and UD12 ROM towers; rather than mess around with soldering close to the chips, I cut these wires and spliced another piece in. Naturally, all connections were resoldered and covered with insulating tape. Finally I replaced the boards, fastened the switches and stuck on the labels. With a sigh of relief, I found that everything worked just fine.

Is this the end of the story? Well, not quite. On page 372 of CBM Professional Computer Guide, A. Osborne et. al., are instructions for adding a capability to switch three ways: first, to RESET the computer; second, to accomplish an NMI (Non-Maskable Interrupt); third, to set the Diagnostic Sense of the 6522 VIA either ON or OFF. [Ed. We later discovered that some editions of the Guide do not mention the subject on p. 372; some do. Sigh. Find a copy that does.]

I'll not repeat Osborne's instructions here. This setup is designed for the 6502 side and works in 6809 as well. Let's look at the capabilities one by one: RESET is a warm start for the computer. [You can accomplish the same thing in either 6502 or 6809 modes by flipping the processsor switch (6809-6502) on the right side of SuperPET.] When you RESET, you lose all programs and data; all pointers re-initialize, as if you'd just turned on power. It's the only way to recover from a crash, short of turning the computer off and on again (in 6502 you can use the Diagnostic Sense switch, enter the 6502 monitor, and recover a program).

Second, you can flip the same switch the other direction (it has two) and cause a NMI. In 6502, this flips you into BASIC. In 6809, Diagnostic Sense may be on or off; it doesn't matter. See the chart below; note that you can usually recover a program.

Condition	6502	EDIT	BEDCALC	mFORTRAN	mAPL
Reset D. Sense Off	Warm Restart	----------- Back to Waterloo Menu ---------------			
Reset D. Sense On	Enter Monitor	----------- Back to Waterloo Menu ---------------			
NMI D. Sense On or Off	BASIC Ready	Edit Menu Text OK	BedCalc Menu Text Lost	mFOR Menu Program OK	mAPL Menu Worksp. OK

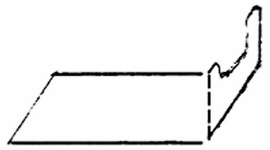
The rig requires two switches: a double pole, center-off job for RESET and NMI, spring-loaded to return to OFF, plus a separate switch for Diagnostic Sense. Though Osborne used connectors, I hard-wired the necessary connections because I have other uses for the user or parallel port. It's easy to solder a wire to the correct lug of the port. If you make the connection close to the circuit board, you have ample clearance to plug in a connector for some peripheral without interference. I also soldered the wires to the J4 and J9 connectors because I did not want to have Osborne's test clips dangling around. The two new switches were installed behind the SuperPET switches I had already moved.

Finally, I added a 5 volt outlet at the back of the computer. 5 volts can be obtained from the correct lug of the cassette port (use a voltmeter to check which one). For this addition, I installed a miniature phone jack by drilling a hole at the back near the other outlets. Again, I soldered the wire so that the cassette port would never be tied up.

These changes were well worth the effort and easy to do considering that I have no real electronics experience. You can leave infinitely looping functions in mFOR and mAPL with NMI; the computer comes back nicely, although a few variables are rattled loose. I tried the same thing many times and finally did manage to scramble a workspace. I strongly recommend that any program or data be saved immediately after recovery with NMI; performance thereafter is not predictable. You should re-examine anything put to disk to be certain it is still correct.

[Ed. We have two comments. First: if you always solder your panty hose instead of the darn wire, there is a no-solder way to fix those right-side switches so they won't snag and flip by accident. On the SPET models with a single switch per mount, pry 'em off with a clean, small putty knife. They're only glued on. Smear a heavy coat of contact cement on the back of each switch and on the computer. Wait twenty minutes. Stick 'em on, rotated 90 degrees, so the toggles face the front of the computer. The toggles won't snag again.

If you have two switches in a single, monolithic mount, this won't work. So get a piece of aluminum gutter sheathing. You can cut it with shears. Make an "L" which lies on its side like this: ______|. The piece must be deep enough (from front to rear of the computer) to completely hide the switches. Slip the aluminum L under SPET until its right front rubber foot rests on and holds the L in place. You can slip a finger inside to use the switches, but you'll have no more accidents. If you care to be fancy, cut an access U in the upright. See the sketch at left.



Second: We sent a copy of this material to guru Terry Peterson, who responded: "You should point out that SPET has the capability to pull the diagnostic sense low under software control (see p. 14 of the Systems Overview Manual) just by setting bit 3 of $EFF8. Now for the bad news: Apparently most 2-board machines don't have this feature included--although it requires only that you install a jumper (pin J3 on top board to pin J9 of bottom board). This was obviously designed-in, but somebody at Commodore decided to save the price of the jumper!"]

THE APL EXPRESS by REG BECK

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

John Koch of Ottawa, Ontario sent along some utility programs for working with directories in APL. You can sort them, split them into two halves, and print them. One of the programs will search through the directory looking for a file and will tell you where it is in the directory. In original form, it would do wildcard searches on prefixes. We added a suffix search capability. Though a bit slow on the search (it's all in APL) it is very handy for those long directory listings which scroll by fast. The program now is menu-driven with help screens. If anyone is interested in a listing of the functions please send me a selfaddressed envelope and a dollar for stamps and copying. An explanation of how to set up the help screens will be included. The program will be included on the next ISPUG utilities disk, but that won't be for some time.

The remainder of this column is devoted to relative files. Most readers should be familiar with simple file terminology. The definitions of a few terms follow for those who are not. The discussion will be restricted to text files.

data:	Characters having meaning.
record:	A collection of related data.
field:	A subsection of a record, typically consisting of one data element such as a name, part number, price, address, etc.
file:	A collection of related records, such as an employee file, an inventory file, etc.
relative format:	Fixed length records. Each record has the same length as the longest record in the file. Each record is numbered and may be accessed by record number in any random order.
key:	A data element which is associated with a record number so that the corresponding record may be accessed in a meaningful way. The data in a specific field is typically used as a key.

Disk space will be wasted when using relative records. This can't be avoided, because each record is padded to be as long as the longest one. In APL, it is convenient to treat each record as a row of a matrix. If fixed field lengths are used, the field divisions in each record will line up in the matrix into groups of columns per field, which simplifies file handling. The entire file can be viewed as a matrix, even if it is too large for main memory. When the file is initially being created records can be filed in batches, using the QUAD APPEND function. If the batch being filed is a matrix, a single QUAD PUT statement will write each row of the matrix as an individual record.

The maximum record length in a relative file is 254 characters. When the file is being created, the number of characters in the longest record is stated as part of the file name. (F:200)TEST,REL would be the correct name for a relative file named TEST which has a fixed record size of 200 characters on drive zero of disk unit 8. Other device designations are inserted between the bracketed prefix and the rest of the file name. (F:200)DISK9/1.TEST,REL is the correct syntax for drive 1 of device #9. If the size is left out of the file name it defaults to 80 characters. (F)TEST,REL is the same as the previous file except that each record is 80 characters long. Because the prefix (byte size) is not displayed on a directory listing, you must record it elsewhere (in the above case, the directory would show TEST only as a RELative file). Additional information, such as record length, number of records in the file, field positions and names can be stored in a sequential file. This file can accessed before any updating of the relative file is attempted and can itself be updated after the user is finished with the relative file.

One often reads that file handling is "not well developed" in APL. What has eluded the critic of APL's ability to handle files is the power of APL's matrix manipulation primitives and idioms. There may be more complex filing functions in other APLs, but in Waterloo APL V1.1 the functions are simple. The hard work of creating, editing and updating the records can be easily handled by matrix manipulation.

The following are simplified examples of functions for manipulating a relative file. The first set are for creating the file, reading it and adding to it.

```
      ∇CREATE[⎕]∇
[  0]    CREATE NAME ;⎕IO
[  1]   ⍝THIS FUNCTION INITIALLY CREATES THE FILE
[  2]   ⍝AND FILES A BATCH OF RECORDS CONTAINED IN MAT.
[  3]   ⍝THE FIRST ELEMENT OF ⍴MAT IS THE NUMBER OF
[  4]   ⍝RECORDS INITIALLY FILED.
[  5]   ⍝THE MAXIMUM RECORD NUMBER IS ONE LESS THAN
[  6]   ⍝THIS SINCE RECORD NUMBERS ARE IN INDEX ORIGIN 0.
[  7]   NAME ⎕CREATE 1
[  8]   MAT ⎕PUT 1
[  9]   ⎕UNTIE 1
      ∇READ[⎕]∇
[  0]    N READ NAME ;⎕IO;A
[  1]   ⍝N IS THE RECORD NUMBER TO BE READ AND IS IN
[  2]   ⍝INDEX ORIGIN 0.
[  3]   ⎕IO←0
[  4]   NAME ⎕TIE 1
[  5]   ⎕SEEK 1,N
[  6]   ⎕GET 1,⍎2↓(A∧¯1↓0,A←≠\NAME∊'()')/NAME
[  7]   ⎕UNTIE 1
      ∇WRITE[⎕]∇
[  0]    N WRITE NAME ;⎕IO
[  1]   ⍝IF N IS ONE GREATER THAN THE LAST RECORD NUMBER
[  2]   ⍝ALREADY IN THE FILE, IN INDEX ORIGIN 0, THIS FUNCTION
[  3]   ⍝APPENDS MORE RECORDS CONTAINED IN MAT TO THE END OF
[  4]   ⍝THE RELATIVE FILE NAMED. IF MAT CONTAINS ONE RECORD
[  5]   ⍝AND N IS A RECORD NUMBER WHICH ALREADY EXISTS, A NEW
[  6]   ⍝RECORD IS WRITTEN TO THAT NUMBER.
[  7]   ⎕IO←0
[  8]   NAME ⎕UPDATE 1
[  9]   ⎕SEEK 1,N
[ 10]   MAT ⎕PUT 1
[ 11]   ⎕UNTIE 1
```

In the above functions, QUAD UPDATE will work for both reading and writing operations. QUAD TIE will only work for read operations. Examples of the syntax follow:

```
NAME←'(F:79)TEST.REL'
MAT←8 79⍴'BOOGALOO'
CREATE NAME            ⍝CREATES THE FILE AND FILES THE 8 ROWS OF MAT AS THE
                       ⍝FIRST 8 RECORDS IN THE FILE.
5 READ NAME            ⍝READS 5TH RECORD (INDEX ORIGIN 0).
MAT←79⍴'X'             ⍝SINGLE ROW TO FILE.
6 WRITE NAME           ⍝FILES THE NEW MAT AS RECORD 6 (INDEX ORIGIN 0).
MAT←4 79⍴'OSHGOSH'     ⍝4 NEW RECORDS AS ROWS OF MAT.
8 WRITE NAME           ⍝FILES THE 4 NEW RECORDS AFTER THE ORIGINAL 8.
                       ⍝REMEMBER THE 8 RECORDS ARE NUMBERED FROM 0 TO 7 IN
                       ⍝INDEX ORIGIN 0 SO THE NEW ONES START AT 8.
```

Each subsequent use of QUAD GET reads the next record. Writing a loop to read a block of records at once is an exercise for the reader. A recursive form should be possible, too.

You must use the entire file name if you wish to erase it from the disk.

```
⎕ERASE '(F:200)TEST,REL'
```

While developing an entire file handling package is beyond the scope of this column, we will examine some of the idioms available and look at a few expressions to see what is possible. Idioms:

```
A∧¯1↓0,A←≠\TEXT∊'()'     ⍝FINDS TEXT BETWEEN BRACKETS.
X∧.=Y                    ⍝COMPARING VECTOR Y WITH ROWS OF ARRAY X.
(A,B)[⍋(ιρA),(ρB)ρN]     ⍝PUT THE VECTOR B INTO THE VECTOR A AT
                         ⍝POSITION N.
T[A,A+(×B-A)×ι|B-A]      ⍝FINDS STRING IN TEXT T BETWEEN INDICES A AND B.
```

The first idiom was used in the function READ to get the record size from the file name. The text between the brackets in a file name could be F:200, for instance. If the first 2 characters are dropped, we have the record size in numerical characters. The execute function changes these characters into the number 200. QUAD GET is followed by a 2-element integer vector. The first element is the tie number specified after QUAD TIE, the second is the record size.

The second idiom may be used to find the record number in a key matrix. The result obtained when this idiom is returned is a boolean vector (zeros and ones) in which the ones are the row positions, where Y is a row of X. As an example, suppose the key matrix had as its rows the last name fields of the file with the record number added, ie:

Smith	000
Jones	001
Rumplestiltskin	002
Aardvark	003
Jones	004
Machinery	005

We have two Jones in the file. When this occurs in a key file they are called synonyms. The file handling program could display all synonyms in sequence, with first names and initials added until the correct one is shown.

If X is the above key matrix (minus the last 3 columns) and Y is Jones suitably padded with spaces, the idiom would return 0 1 0 0 1 0 which could be used to select the 1st and 4th rows (index origin zero, remember) of the key matrix. Picking the last three characters off these two rows and executing them would give us the record numbers. If there were over 999 records (whew!) the record numbers could consist of four characters. In APL:

```
KEY←'JONES'
X←⌽(0,3)↓⌽KEYMAT            ⍝REMOVES LAST 3 COLUMNS OF THE KEY MATRIX.
Y←KEY,((¯1↑ρX)-ρKEY)ρ' '    ⍝PADS OUT THE KEY WITH PROPER NUMBER OF SPACES.
Z←(X∧.=Y)⌿KEYMAT            ⍝SELECTS 1ST AND 4TH ROWS OF KEYMAT.
Z←(0,19)↓Z                  ⍝FINDS RECORD NUMBERS AS A MATRIX.
⍎,Z,' '                     ⍝OBTAINS THE RECORD NUMBERS.
1 4
YMAT          ⍝SELECTS 1ST AND 4TH ROWS OF KEYMAT.
Z←(0,19)↓Z                  ⍝FINDS RECORD NUMBERS AS A MATRIX.
⍎,Z,' '                     ⍝OBTAINS THE RECORD NUMBERS.
```

These expressions are combined in the function MATCH which follows. If the key matrix, KEYMAT, is in the workspace, MATCH will return the record number or numbers if there is a match. If the key is not in the key matrix MATCH returns -1.

```
      ∇MATCH[⎕]
[  0]    R ← MATCH KEY ;X;Y;Z
[  1]   X←⌽(0,3)↓⌽KEYMAT
[  2]   Y←KEY,((¯1↑⍴X)-⍴,KEY)⍴' '
[  3]   Z←(0,19)↓(X∧.=Y)⌿KEYMAT
[  4]   →((1↑⍴Z)=0)/NOMATCH
[  5]   R←⍕,Z,' '
[  6]   →0
[  7]   NOMATCH:R←¯1
[  8]   ∇
```

The last two idioms can be used to insert new fields into old records and to extract fields from records.

* * *

Some SuperPET users may have taken advantage of bargain prices for the Commodore SFD-1001 drive. This drive is essentially half an 8250 and offers a megabyte of storage on both sides of a single disk. The full megabyte can be used to store up to 65535 relative records in a single file. The SFD-1001 is read/write compatible with the 8050, with some reservations. To read or write 8050-formatted relative files in an SFD-1001 drive, the "expanded relative file" feature of the SFD-1001 drive must be disabled. The two functions DISERF and ENERF will disable and enable this feature.

```
      ∇DISERF[⎕]∇
[  0]    DISERF N ;⎕IO
[  1]    ⍝N IS THE DISK UNIT NUMBER (IE; 8 OR 9)
[  2]   ('IEEE',(⍕N),'+15') ⎕CREATE 1 +⎕IO←0
[  3]   ('|+ω',⎕AV[164 67 1 255])⎕PUT 1
[  4]   ⎕UNTIE 1
      ∇ENERF[⎕]∇
[  0]    ENERF N ;⎕IO
[  1]   ⍝N IS THE DISK UNIT NUMBER (IE; 8 OR 9)
[  2]   ('IEEE',(⍕N),'+15') ⎕CREATE 1 +⎕IO←0
[  3]   ('|+ω',⎕AV[164 67 1 0]) ⎕PUT 1
[  4]   ⎕UNTIE 1
```

The "expanded relative file feature" is automatically enabled on power up and may also be re-enabled by powering down or resetting the drive.

Appendix E on page E1 of the Disk System User Reference Guide discusses an error in relative files in all Commodore drives. A file can be corrupted if a particular program sequence is used to update files. If you have problems with relative files, this may be the cause.

TALKING TO YOUR PET VAX
by Stan Brockman
Associate Editor

[Ed. We recently received a clever program from some SPET folks in Canada who were having trouble uploading files from a SuperPET to a VAX. We'd had reports from Stan Brockman of similar problems, so we asked him for a summary on telecommunications between SPET and VAX, which follows.]

I routinely upload and download files between my SuperPET and a VAX, using the NEWTERM program on the ISPUG master telecommunications disk, with a VAX running VAX/VMS Version 4. The program translates the linefeeds (ASCII $0A) received from the VAX into nulls by default. I did only one thing to tailor NEWTERM for the work: changed the outgoing translation table in NEWTERM so that SPET's code for the DELETE key ($04) is sent to the VAX as ASCII DEL ($7F). This makes it simple for me to correct my typos. You do this quite easily in the monitor by changing the byte at $7138 (in the translation table) from $04 to $7F.

When communicating, I enable Full Duplex (Half Duplex is the default) and employ the default upper-lower case mode. These settings and the revised translation table in NEWTERM can then be saved to disk in a file configured especially for TC with a VAX (I call mine VAXTERM), using the command built into NEWTERM for that purpose. Once you have the program configured, all that's left is to set the baud rate, call the VAX, and log in.

Downloading files from the VAX to the SPET is fairly straightforward, except that a few unwanted blank lines are put into the file. The procedure is to enter the VAX command, "TYPE file_name", but before hitting <RETURN> to press PF8 and answer the prompt with the name of the file under which SPET is to save the received file. Then you press <RETURN> to start the listing from the VAX. You use PF. to close the file after it has been received. You can edit out the unwanted blank lines later. (Does anybody know of a way to avoid the blanks?)

Uploading files to the VAX has been a problem for me and for others. Except for the shortest of files, the VAX always aborted the process of storing the transmitted data; it detected an "Output Buffer Full" error. I was frustrated for months tryng to convey the essence of my problem to our System Manager and then uselessly trying his suggestions. Eventually, my persistence paid off. We found that if the VAX's echo is shut off, the entire file is received and stored. It isn't yet clear why the error occurred, but it seems that the VAX output buffer overflowed while trying to transmit more characters than it had received.

For example, the VAX echoes each <RETURN> with a <RETURN> and a linefeed. At baud rates of 150 and above, it was unable to squeeze the extra characters into the return (echoed) transmission to SuperPET. The output buffer is not capacious on our VAX; it fills quickly and generates the error.

So, what is the upload procedure? Prior to transmitting an ASCII file to a VAX, execute the VAX command, "SET TERMINAL/NOECHO". Because the VAX will no longer echo, press PF7 to switch SPET to local echo, so you can see what you are typing. Then get the VAX ready to store the file you are about to send by entering "CREATE file_name" <RETURN>, hit PF9 on the SPET, and answer the NEWTERM prompt with the name of the file to be sent. You can now rewarm your coffee while SPET and NEWTERM do the rest. When the file has been transmitted, you send VAX a CONTROL Z to close its file. The result should be a file without extra blank lines stored on the VAX. The procedure has worked well for me.

I've checked the SET TERMINAL/NOECHO procedure above at speeds up to 1200 baud, with no problems, and have successfully transferred data with NEWTERM at 4800 baud on a direct connection with the echo so shut off.

I'd appreciate hearing of simple procedures used by others, especially *if there* is one which avoids the blank lines on downloads. I've heard a rumor of the existence of KERMIT, running on SuperPET... Can anyone confirm this or let me know how to get it?

WINDOW INTO OS9
by Gerry Gold and Avy Moise

You may have wondered why an 8050 REL file may hold only 160-178K of data in OS9. You also must have noticed that all OS9's RELATIVE files are named 'OS9 DRIVE A'. Most CBM 8050 drives cannot handle RELATIVE files larger than 178K. We realize that this restriction may become bothersome, so as a temporary solution the user can build two logical file system on each 8050 drive, thus extending the storage capacity of an 8050 single drive to about 350K.

On the 'Super-OS/9 B09+MOD' disk, you will find two files, 'd8d8' and 'd8d9'. These files represent two different configurations of the OS9 LOAD/BOOT module, which is found on your SYSTEM disk (default is OS9=d8d8). 'd8d9' lets you access four logical drives, resident on two CBM dual drives, addressed as units 8 and 9. 'd8d8' will allow access to four logical drives resident on one CBM dual drive system, addressed as unit 8. The first RELATIVE file system on each drive is referred to as 'OS9 DRIVE A'. The second file system on each drive is known as 'OS9 DRIVE B'. In the release version, we have copied 'd8d8' into OS9; it therefore will support two logical drives on each physical disk:

disk8/0.os9 =	d8d8	d8d9
OS-9 name	Waterloo OS name	Waterloo OS name
/D0	(f:129)disk8/0.OS9 DRIVE A,rel	(f:129)disk8/0.OS9 DRIVE A,rel
/D1	(f:129)disk8/1.OS9 DRIVE A,rel	(f:129)disk8/1.OS9 DRIVE A,rel
/D2	(f:129)disk8/0.OS9 DRIVE B,rel	(f:129)disk9/0.OS9 DRIVE A,rel
/D3	(f:129)disk8/1.OS9 DRIVE B,rel	(f:129)disk9/1.OS9 DRIVE A,rel

OS9: BACKUP This command is similar to the CBM DOS command 'C1=0' or 'C0=1'. The main similarity is the fact that in both systems the floppy disks must be identically formatted before the copy command may proceed. In OS-9, you must create a dummy file system (using the FORMAT utility) with attributes that exactly match those of the floppy which you wish to backup (i.e. same number of cylinders, density and sides). Then and only then will the command succeed.

COMPUTER FOR SALE Emory B. Antonucci of 801 North Ashton Street, Alexandria, VA 22312 (703 354 8495) used his SuperPET at work and, having retired, wants to sell it and his 8050 drive, language and tutorial disks, all Waterloo language manuals (excepting COBOL), all CBM reference manuals, together with installed WordPro and InfoPro ROMs and manuals. His 3-board SPET and drive are in operating condition. Write or call. Asking price is $750 for the package.

SPET with CP/M for Sale Gary Hermann of 16906 Big Falls Road, Monkton MD 21111 (301 357 8016 after 6 p.m. on weekdays) offers for sale at $1200 a SPET with an 8050 drive, ADA 1800 interface, and SSE CP/M system, Borland's Turbo Pascal, Nevada CP/M editor, and other miscellaneous software.

LANGUAGE COVERAGE ONCE AGAIN LIGHTLY Only two languages in SuperPET aren't covered by hundreds of books--microBASIC and Waterloo's Assembly Language. We therefore cover both intensively because you have no other source of information. We do make an exception for APL because of the its implementation on SuperPET, and SPET's small workspace.

We're happy to cover mFORTRAN, mPASCAL, or mCOBOL if there are bugs in languages as found in SuperPET, or tips which are peculiar to using that language on SPET. But space won't let us print lots of programs in all languages--there's no room.

We'd love to get programs in any language which are unusal, powerful and short, particularly if they hook into SuperPET's library or operating system and demonstrate how to use that language as implemented in SuperPET. But please document what you send in--there are hundreds of you and only one editor--and there's no time to guess at the intent of what you send. Twenty minutes per letter does ye ed have, and if in that twenty minutes it isn't clear how to use what you send, we have to pass on to the next letter or disk. The ratio of readers and writers to editors in this organization is not mano e mano!

We haven't had much in the way of contributions from people who use mPASCAL or mCOBOL, which is bad. We doubt the implementations on SuperPET are bug-free, and we suspect there are some good ways to bypass problems with those languages in SuperPET. Why not share your knowledge or confess your ignorances?

BOOBOOS IN BOOLEAN In several SPET languages you can state boolean relations improperly without a syntax error but have your program itself bomb. We show an example below. If you try to parse the example, you'll get nowhere. You must use parentheses, as in the second example. If you leave the parens out, your results can be strange indeed. Be careful with "if var", for, dedepending upon language, it may mean "if > 0" or it might mean "if not 0". Variables can, of course, hold negative values....

```
 if variable_1 or variable_2 and variable_3
   do thus and so  [Fails]

 if (variable_1 or variable_2) and variable_3
   [Runs okay. Shift parens; meaning changes.]
```

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 7: $3.75 each.

Send check to the Editor, PO Box 411, Hatteras, N.C. 27943. Add 30% to prices above for additional postage if outside North America. Make checks to ISPUG.

===

DUES IN U.S. $$ DOLLARS U.S. $$ U.S. $$ DOLLARS U.S. $$ U.S. DOLLARS $$

APPLICATION FOR MEMBERSHIP, INTERNATIONAL SUPERPET USERS' GROUP

(A non-profit organization of SuperPET Users)

[] Check if you're renewing; clip and mail this form with address label on the reverse side. If you send the label, don't fill in the form below.

Name:______________________ Disk Drive: ________ Printer:__________

Address:____
Street, PO Box City or Town State/Province/Country Postal ID#

For **Canada** and the U.S.: Enclose Annual Dues of $15:00 (U.S.) by check payable to ISPUG in U.S. Dollars. DUES ELSEWHERE: $25 U.S. Mail to: ISPUG, PO Box 411, Hatteras, N.C. 27943, USA. **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. Please mail all inquiries, manuscripts, and applications for membership to Dick Barnes, Editor, PO Box 411, Hatteras, N.C. 27943. SuperPET is a trademark of Commodore Business Machines, Inc.; WordPro, that of Professional Software, Inc. Contents of this issue copyrighted by ISPUG, 1985, except as otherwise shown; excerpts may be reprinted for review or information if the source is quoted. TPUG and members of ISPUG may copy any material. Send appropriate postpaid reply envelopes with inquiries and submissions. Canadians: enclose Canadian dimes or quarters for postage. The Gazette comes with membership in ISPUG.

ASSOCIATE EDITORS

Terry Peterson, 8628 Edgehill Court, El Cerrito, California 94530
Gary L. Ratliff, Sr., 215 Pemberton Drive, Pearl, Mississippi 39208
Stanley Brockman, 11715 West 33rd Place, Wheat Ridge, Colorado 80033
Loch H. Rose, 102 Fresh Pond Parkway, Cambridge, Massachusetts 02138
Reginald Beck, Box 16, Glen Drive, Fox Mountain, RR#2, B.C., Canada V2G 2P2
John D. Frost, 7722 Fauntleroy Way, S.W., Seattle, Washington 98136

Table of Contents, Issue 7, 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.