If we are going to get compilers for the SuperPET languages, it seems obvious we're going to have to write them ourselves. So, let's be about it. John Toebes, resident genius, has offered to start with a compiler for microBASIC--if he can get some help in disassembling microBASIC first. He could do that by himself, but it will take too much time. So we ask those of you who can handle assembly language to pitch in and help. Send your name to the Editor, at PO Box 411, Hatteras, N.C. 27943. We'll put John in touch with you, and arrange to break the job up into small pieces, so no one is overloaded. The rest of the story goes like this: no volunteers, no dissassembly, no compiler. (See separate note, this issue, on a PASCAL compiler.)

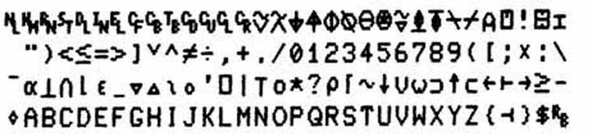
◇◇

ARE YOU REDMARKED?

Look at the mailing label on this issue. If, underlined in red, you find a note that your membership has expired, this is your last issue. Please renew now if you're going to renew at all. Check the 'renew' block on the form, last page, and either mail the form with your address label or send us a copy.

◇◇

JACKPOT!! APL CHARACTER SET FOR COMMODORE 8023 PRINTERS Some 30% of all ISPUGgers use the 8023, and we have long hoped that some creative genius would develop a way to print the APL character set on that printer. Well, Delton P. Richardson of 4199 Old Bridge Lane, Norcross, GA, 30092, has done it. Delton reports: "I use Steve Zeller's bit-mapped graphics method, and use the same 8x10 characters, and his support, character definition, and dump functions. I am very pleased (with the help of the Gazette) to have an APL printer... My only problem is that I don't know how to avoid sending a carriage return within one function, which causes a few dots to be printed at the very end of each line--a minor inconvenience." Printed below is a reproduction of Delton's small character set as it comes off his 8023. We suspect his routine might handle the 4023, but from tests we know it will not handle the 4022. Those who want a copy of his character set and of all functions can get it on disk. We have added his material to the ISPUG APL Character Set disk announced last issue.



Order 8050 format from Editor, PO Box 411, Hatteras, N.C. 27943, or 4040 from Secretary, 4782 Boston Post Road, Pelham, N.Y. 10803. Send $10 U.S. to ISPUG. Those of you who already have the disk: send it back with return postage and a good mailer. We'll add Delton's stuff and return the disk to you. The character set printed above is small to save space; Delton's WS also prints a larger set. The quality of the original is excellent; it doesn't reproduce as well above. Gee, all we need now for the full and happy life is a spreadsheet for SuperPET which employs all of the upper 64. Anybody up to writing one?

LATE FLASH Some folks who have the FX-80 printer and who got the ADA 1800 interface are having problems getting 5 volts from the printer to the 1800. We have a note from Reg Beck on precisely how to do it. Send a SASE. We'll send back a copy of Reg's note, which we've also put on the APL Character Set disk.

◇◇

PATCHES! PATCHES! mBASIC, mFORTRAN Prof. Dr. Friedrich Stummell of Johann Wolfgang Goethe University in Frankfurt sent us a note about a bug in V1.1 mFORTRAN, for which Waterloo had issued a corrective patch; in some programs, a small array was treated as having more di-

```
 40 ! microFORTRAN patch: fort_patch2:bp
 50 dim a%(43)
 60 data 5,421,231,100,106,100,45,20,236,228
 70 data 52,6,236,248,4,189,176,96,237,228
 80 data 174,98,48,2,175,98,32,232,-1
 90 data 69,408,29,-1,69,445,0,193,10,-1
100 data 73,250,247,-1,-1
110 open #2, "disk/1.FORTRAN,PRG", input
120 open #3, "disk/0.FORTRAN,PRG", output
130 ! End of preamble. Remainder is
140 ! same for mBASIC and mFORTRAN patches.
150 x=peek(86)*256+peek(87)+4
160 y=peek(x)*256+peek(x+1)+1
170 poke y,0,0
180 mat read a%
190 i%=0 : j%=1 : k%=1 : l%=1
200 loop
210   for j%=j% to a%(i%)
220     for k%=k% to 512
230       get #2,x%
240       print #3,chr$(x%); ! SEMICOLON
250     next k%
260     k%=l%
270   next j%
280   i%=i%+l%
290   for k%=k% to a%(i%)
300     get #2,x%
310     print #3,chr$(x%);    ! SEMICOLON
320   next k%
330   i%=i%+l%
340   loop
350     get #2,x%
360     print #3,chr$(a%(i%)); ! SEMICOLON
370     i%=i%+l%
380     k%=k%+l%
390   until (a%(i%)=-l%)
400   i%=i%+l% : j%=j%+int(k%/513) : k%=mod(k%,513)
410   if k%=0 then k%=l%
420 until (a%(i%)=-l%)
430 on eof ignore
440 loop
450   get #2,x%
460   if io_status=2 then quit
470   print #3,chr$(x%);       ! SEMICOLON
480 endloop
490 close #2 : close #3
500 stop
```

mensions than it really had; the interpreter stopped execution and issued an error message. We were about to print the patch which corrected that error when info-WAT arrived with a new mFORTRAN patch (see left). The new patch not only corrects the array error noted above but also gets rid of the display bug noted in issue 11 on page 169 (factor of 10 error in large decimal displays). We've patched mFORTRAN and have tested a bit; the bugs seem to be gone. Below is a test program from Dr. Stummell which shows the bug in unpatched V1.1. The test program runs okay after you patch.

```
program test_mfortran
real b(5,9)

do i=1,5
  do k=1,9
    b(i,k)=10*i+k
  enddo
enddo
call srwrite(b(1,1),45)
end

subroutine srwrite(x,n)
real x(n)
print
do i=1,n
  print,x(i)
enddo
end
```

Unpatched, V1.1 treats 'b' as though it had 5x5 dimensions. Please don't rush to patch, but read on. You just might create a horrid mess.

First, DO NOT patch V1.0. It won't work. Patch V1.1 only.

Second, V1.1 mFORTRAN, as recently sent to some by ISPUG, is not patched. We shipped the disks as received, and did not have the patch at that time. You get the pleasure of copying the patch and waiting 39 minutes for it to run (instructions on how to patch, below). Warning: the first ten ISPUG Utility disks shipped contain a "fort_patch:bp", which is the early patch issued by Waterloo to cure the matrix problem only . If the filename on the disk shows "fort_patch2:bp", you've got the one above. We suspect no harm'll be done if you

run patch2, above, after running plain "patch", but strongly recommend you patch straight V1.1 mFORTRAN (and not the version you've already patched), just to be on the safe side. Only ten of you face the problem. Last, don't scratch old V1.1. Who knows when another patch may come along, or if we'll later find a bug?

MBASIC NOW CHAINS! The same issue of infoWAT prints a patch for mBASIC which rids it of its inability to pass array (awright, matrix!) values when chaining.

```
 40 ! microBASIC patch: mbasic_patch3:bp
 50 dim a%(33)
 60 data 24,30,39,-1
 70 data 26,182,0,-1
 80 data 54,40,221,-1
 90 data 55,346,158,70,48,1,18,159,72,158,72
100 data 159,74,111,132,48,6,18,159,76,-1,-1
110 open #2, "disk/1.BASIC,PRG", input
120 open #3, "disk/0.BASIC,PRG", output
130 ! Copy mFORTRAN patch from line
140 ! 150 to the end.
```

The program, from line 150 to the end, is IDENTICAL to the mFORTRAN patch above (from line 150 to the end). In different words, delete lines 40-140 from the mFORTRAN patch above, and substitute the lines at the immediate left. We took both programs straight from disk after we'd patched and had tested, so there shouldn't be any copy errors. Yes, mBASIC CHAINS matrix values. For a test program see p. 122, No. 9, which we expanded for larger matrices and more of 'em, and sent to printer, terminal, and to disk--all okay. But tests were limited by time, so don't scratch old V1.1, whether or not patched with "mbasic-patch2", until experience shows this latest version is clean and serviceable.

Don't rush to patch; you might cause problems. Instead, sort out who has done what, with which, to whom: First, there are three existing patches for mBASIC:

mbasic_patch1	Bug. Don't use.
mbasic_patch2	Use alone.
mbasic_patch3	Use alone.

Patch1 was published in Vol.1, No. 5. p. 32. It's buggy. DO NOT USE IT. Patch2 is found in Vol. I, No. 6, p. 38, and cures the problem of a CR after 79 characters. All V1.1 disks recently sent out by ISPUG are patched with patch2. Patch3 is the patch printed above. It may be run on straight V1.1 microBASIC, as issued by Commodore, or on mBASIC patched with patch2. DO NOT use any of the three patches on V1.0 mBASIC. Okay, that's not easy to follow, so we print below a summary of what flies and what the wings come off of.

Flies:	Flies:	Flies:	Superthud:
V1.1 as issued: patched with:	V1.1 from ISPUG (has mbasic_patch2)	V1.1 you patched with mbasic_patch2:	Any V1.0 with any patch, or V1.1 with patch1
mbasic_patch2 mbasic_patch3	add mbasic_patch3 (mBASIC patches in about 31 minutes, mFORTRAN in 39 or so.)	add mbasic_patch3	

Whew. In copying, WATCH THOSE SEMICOLONS! Proof carefully. Since some APL and PASCAL types are illiterate in microBASIC, but everybody can use the mED, here is how to enter, save, and run the patches and to how to get a new language disk with the modified languages on it, in one pass, with one COPY operation.

First, load mBASIC; then enter the mED with: edit <RETURN>. Enter the patches. When proofed, save 'em to a good disk with: p mbasic_patch3:bp (or whatever) <RETURN>. The disk the patches are on we now call the PATCH disk. Take it out.

Then load in drive 1 the OLD V1.1 language disk or a good copy; protect it with a no-write tab. Put a new disk in drive 0; copy disk 1 to it. You have two ways:

1) if your drives will BACKUP with: g ieee8-15.D0=1 given at command cursor, do it that way. But if your drives belch READ ERROR on backups, then, 2) NEW disk 0 with: g ieee8-15.N0:lan_patched,lp at command cursor. Then copy disk 1 to disk 0 with: g ieee8-15:C0=1. Either way, you now have a copy of the old V1.1 language disk on drive 0--call it the NEW LANGUAGE disk. Take it out, and stick the PATCH disk in drive 0.

Load a patch into mED in microBASIC with: g fort_patch2:bp (or whatever). Take out the PATCH disk and stick the NEW LANGUAGE disk back in drive 0. Then hit SHIFT/RUN. The mED will transfer control to mBASIC; the patch runs. The new, patched version of the language will overwrite the old version in drive 0. When the patch program is finished, repeat the process for the other language. You now have a completely patched, new language disk in drive 0, and you haven't had to copy any single files, one by (yawn) one--including all those neat ML modules on your old language disk, and all those .exp files from Waterloo...

Then test the hell out of the language versions on the NEW language disk. If you have problems, proofread the patches, correct 'em, and run 'em again. When the patched languages run okay, backup the NEW language disk. Save old V1.1, well and truly labelled, for who knows when we might have to go back to it.... Bugs have more lives than cats.

Sorry for all the nitty-gritty, but we have a lot of new members out there, and know some folks never did figure out how to run the patches published before. So grin and bear up, you old hands.

◇◇◇

A TERMINAL EMULATOR FOR APL and GENERAL USE

Do you want terminal emulator software which can handle general telecommunications in SuperPET and which is "the best APL terminal emulator available," in Steve Zeller's words? See Steve's column this issue. For background, we reviewed COM-MASTER in issue 10, p. 136. You'll find more background in Zeller's review of the Waterloo Terminal Emulator on p. 139, issue 10. When you've read that material, see what Steve says in his column, this issue, about the capabilities of the current version of COM-MASTER. Apparently all the hard work put in by Dan Jeffers (author of COM-MASTER) to give his package a full APL capability has paid off handsomely. We owe Steve Zeller thanks for patiently evaluating the work as it progressed. To Dan Jeffers, kudos for persistence in writing the only telecom package we know of which will handle all SuperPET files in all SuperPET languages. Don't overlook this software. It was good before; now it's a whiz.

◇◇◇

ONCE OVER LIGHTLY
Miscellany

So you want to 'move text' in the microEDITOR without any previous save to disk, Jack? John Toebes of Raleigh, our resident genius, told us how. First few times we tried it, we crashed (so be warned), but after we practiced, it worked. Suppose you want to copy this line of text to the end of this note, and the '#' command tells you that the last line is number 17. We copy the line above to line 18 (right after line 17, right?) with the following steps: 1) At command cursor, enter 17 g terminal <RETURN>. The cursor will disappear; touch HOME to get it back. Then, 2) Put the screen cursor on the line you want copied, 3) Hit RETURN and fast as scat hit STOP. The line indexed by the screen cursor is duplicated as line 18. Someone in back cries; "You said: 17 g terminal, and you duplicate on line 18!" Just so. The command copies to the line after line 17, just as a 'get'from disk does. The secret is to hit STOP a nanosecond after you hit RETURN (awright, a millisecond...). You can duplicate a batch of lines if you hit RETURN a number

of times, and touch STOP instahootly after the last RETURN. Try it on a scratch file. Jack be nimble, Jill be quick.

DIOGENES, STOP LOOKING Diogenes roamed the streets of ancient Athens, lamp lit, endlessly searching for an honest man. Blow out the lamp, Diog--we found one. A few months back, we shipped a Master Telecom disk to The Computer Shop of Calgary, Ltd., 3515 - 18th St., S.W., Calgary, Alberta T2T 4T9. In today's mail, with no warning, back came a check for $45, for ROYALTIES on copies made by the Computer Shop for SPET owners. In an age of disk piracy, an honest dealer! It restoreth our waning faith in mankind.

A RIP-OFF? Somewhere up in Canada is a highly upset ISPUGger who ordered the V1.1 software and manual updates, and sent a check with only the bank's name on it. Either the Secretary or the Editor lost the envelope/letter in which it came and we haven't shipped because we don't know where to send the stuff. Peace. If you'll identify yourself (send the name of the bank on which the check's drawn), we'll send V1.1.

OOPS & AHEM DEPT. We lost a line from Loch Rose's article in issue 12, page 196 (line 7 is AWOL). Add line 7 to your copy: "not matter for now.) When this is done, the variable 'dy_' will contain the val-".

Loch also notes (just after getting V1.1 software) that some of his previously published articles on relative files must be revised to work in V1.1, to wit: p.80, top page, line 30: add comma after a$; p.171, Program 2, add commas at end of lines 230 and 270; in Program 4, add commas at end of lines 620 and 650. Make notes on those pages. Loch adds you may need commas on lines 60 and 70 on page 172 as well. He has not checked. We used V1.0 to run his programs, but neglected to mention the programs were in V1.0. Henceforth, on all submissions, state VERSION so ye ed doesn't forget--again. Last, Loch notes we called the location of memend_ as $32 on p. 187, line 10. Change it to $22! Meanwhile, we will garrotte the proofreader--if we can manage to creep up behind ourself.

SECONDARY ADDRESSES AND EXPANDED PRINTING Nicholas Kaps of 378 Sheridan Court in Manteo, CA 95336 told us how he transmits secondary addresses to Commodore printers. On his 2022, you can send any of the secondary addresses in the format at the left (where x stands for the secondary address). We already knew you could send secondary addresses to IEEE4 printers with the the second line at left. Nick also notes he found a way to get expanded printing from his 2022 whilst in the mED: "Enter text you want printed in expanded format pre-fixed by two different characters such as 'ab', like this: ab this is a test. Go into the monitor and start a dump at $0a00 until you find your text:

```
printer4-x
ieee4-4
```

```
Revision:          01 01
ASCII Code in Hex: 61 62 20 74 68 69 73 20 69 73 20 61 20 74 65 73 74
Text:               a  b    t  h  i  s    i  s    a    t  e  s  t
```

"Then change the 61 and 62 above to 01 01 (see revision line above); hit RETURN. Leave the monitor. When you print the file, the test line will appear at printer in expanded format. You will no longer be able to access the area of the file preceding your change. Do not save the file to disk!" Nick sent samples to prove it. Those who read Gary Ratliff's article on text compression in the mED last issue will understand why a CONTROL code mucks up the mED file. Flash: see separate note, this issue, on sending printer control codes from the mED!

A COMPILER FOR PASCAL In came a note from Barry Bogart: "Just got a product called 'Zoom PASCAL'...although I am not an expert on PASCAL it seems about the same functionally as our Waterloo version. At least the omissions from 'standard' PASCAL are about the same, and ZOOM has some extensions from UCSD that our PASCAL lacks. The really nice thing about it is that it accepts ASCII files! So I can use SPET to develop and debug, and ZOOM on the C64 to compile. The PASCALs are so close that all the ZOOM examples run on SPET and all the 'PEX' examples I tried compile on the 64! The difference is that the comments found as program parameters in the ZOOM examples must be deleted. But does it run on the 8032? Unfortunately not. But that is probably because of very minor mapping differences. If there is ever an 8032 version of it, we should have our first compiler." Barry notes some speed differences: Waterloo interpreter using the Sieve of Eratosthenes: 4 minutes for all primes to 1000. ZOOM compiled: 4.48 seconds. ZOOM cost Barry $61 in Canada, and includes a P-code compiler and N-code translator.

ACCOUNTS RECEIVABLE? Anybody have an Accounts Receivable program which will handle up to 500 accounts and will work on a SPET with an 8250 and a 4023 printer, preferably in mCOBOL? If so, write: George Parry, 34 Bellefontaine St., in Agincourt, Ontario, Canada M1S 1J7. Suspect he'd take it in Swahili if it works.

AGRICULTURAL SOFTWARE Phil Cameron, Director, Computer Services at Lakeland College, Vermilion Campus, Vermilion Alberta T0B 4M0, has almost 90 SuperPETs and much agricultural software. He sent a directory of all such software, for all current computers (Radio Shack to Apples and the 8032). It's 37 pages long. If you want a copy, send $5 to ye ed for copying and postage at Box 411, Hatteras, N.C. 27943.

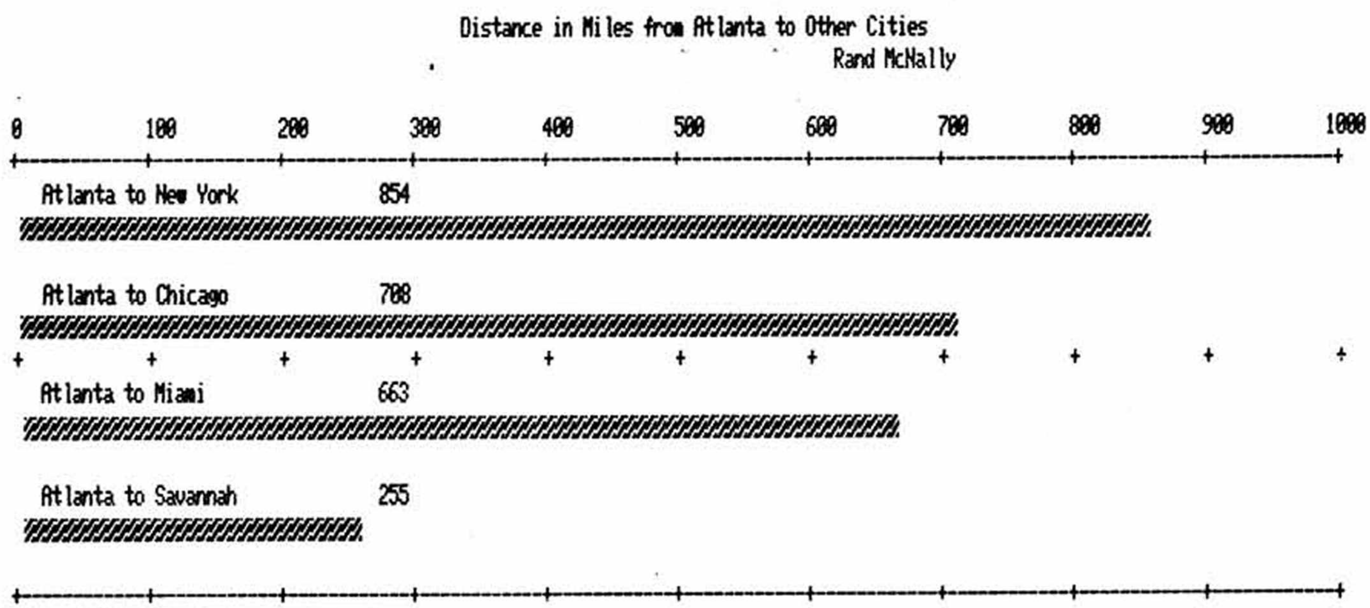
PHOTOGRAPHING THE SCREEN Dr. Jakob Bennema of Bennema Agriculture University, Wageningen, Netherlands, gets in one step a negative film (similar to a diapositive) so he can project for instruction large images of SPET screens. On projection, he gets black characters on a white field. His sample is bold and clear. If you try this, he recommends a reflex camera with manual controls (and notes that "if you have an automatic, buy a cheaper one!"). He says 1) use the reflex viewer to come close enough to the screen to see the screen only; 2) remember that the screen is <u>not</u> flat, so that a wide-open lens may give you too little depth of field and an out-of-focus picture, 3) exposure is determined by the brightness of one character, not the number of characters (which is why the fancy automatics fail), 4) to get an idea of proper exposure, fill the screen with reverse-field spaces (ASCII 32+128) and take test exposures. With a panchromatic film of 360 ASA he has the best results with a setting between f:8 and f:10 at .5 seconds exposure (he hasn't tried other films). From our own experience, we'd try high-contrast copy films (those used for microfilming).

FOR DEVICE X... We had an inquiry on how to send DOS commands to a second set of drives (in this case, device 9). Well, if g ieee8-15. addresses device 8, channel 15, it seems logical that g ieee9-15. will address device 9. And so it does. Anything you can do with the first command you can do with the second.

QUICK HENRY, MORE FLIT Two more bugs found in SuperPET: 1) You cannot load a file from disk/1 in the monitor if the filename is longer than 13 characters, for the monitor refuses the 'load' with an 'invalid command' error. The same command, with a full 16-character filename, is accepted for drive 0; and 2) the command to copy a directory to disk fails quite consistently on some disks which are about half-full. Though part of the directory copies to the new disk file,

the BLOCKS FREE are in serious error, and from two to five files don't appear on the new list. So, if you copy a directory with 'di disk dir', better check the file 'dir' to make sure all the disk files are listed. The error always occurs at the very end of the file. Tests made in V1.1 only, on three separate disks. We have enough bugs in SuperPET software at this point to warrant the issue of de-bugged software. How about it, Waterloo? You, too, Commodore!

BAR GRAPHS ON DEMAND Delton B. Richardson of 4299 Old Bridge Lane, Norcross, Georgia 30092, wrote a beautiful package which creates bar-graphs large or small upon demand. It automatically adjusts scale to accomodate the values entered, and uses the graphics capabilities of Commodore printers (or of any other printers, since you can modify the program easily) to make attractive graphs, of which we print a sample below. Delton sent a batch; you see one of the small ones. Ten minutes after loading this program, we printed our own bar graphs. Delton provides a tutorial and full instructions, which we put on the ISPUG Utility disk we define this issue. Delton also added the tutorial and instruction to his Home Accounting package (Vol. I, page 146). This program, like Home Accounting, is professionally done, menu-driven, and well-explained. Yes, you can change margins, and create big graphs on wide printers or narrow graphs with condensed printing, if you prefer.



It is easy to change the characters which form the bars; with one keystroke we changed to ASCII 124 '¦' on DIABLO for our own bar graphs. To save space, we limited Delton to four bars. His program will do 27, and at widths ranging from less than half of the graph above to the full width of an 8023's printing area. You can form the bars from any graphics character your printer can output.

MODULO WHAT?? : ABSOLUTE CURSOR CONTROL MARGIN CONTROL, SPLIT SCREENS AND SUCH Unlike previous Commodore machines, SuperPET provides absolute cursor control in all languages, though at first glance only microBASIC possesses this capability. You'll find the cursor position reported in two bytes, starting at $122. If you look there, you'll find the cursor value ranges from 1 (Home position) to 2000 (end of line 25). Explore it in the monitor, and try to define exactly what position is reported at $122. (If you change the monitor dump to start at $120, you may get a clue.)

Surprisingly enough, microBASIC is the only language which uses the 1-2000 range at $122 directly. In all the rest, the interpreters handle cursor position modulo 256. For example, the first time we used library function TGETCURS_ in mFORTRAN, we thought it'd return the cursor position by row and column, just as the Assembler manual tells you (it says high byte is row and low byte is column). But the first number we got was 4383--and there aren't 43 rows or 83 columns on SuperPET's screen. Then we remembered an APL program from Jim Swift, which treated the return from TGETCURS_ modulo 256, and everything clicked into place. The value of 4383 converts to row and column right easily--modulo 256.

Somebody moans: 'modulo what????' Moan not. It is easy. The modulus is simply what's left over when a number is divided by another; i.e. 4 mod 2=0 (nothing is left over). 5 mod 2=1 (one left over after first division); 100 mod 80=20, and 90 mod 80=10. Row and column on SuperPET's screen are determined first by integer division; then by the modulus. Let's take the value of 4383, as given above for mFORTRAN. Integer divide it by 256. You get 17 (the row). Multiply 256 times 17. You get 4352. Subtract that from 4383, and you get 31 (the column). In a program, you have an easier way. Every language has a function 'mod', however stated. Use it. Column=mod(value,256). Try it with 4383; you get column 31.

After all this, you say: 'Fine. But what do I use this for?' The answer: to obtain extremely powerful, swift and flexible screen control. Want areas the cursor can't enter? Want to limit printing to part of the screen? Want to set margins top, bottom, left or right? Want to sense cursor position and have events occur at specific locations? Want split screens? You may have them all.

Before we get to examples, let's distinguish between microBASIC and the rest of the languages. You cannot make a direct SYS call to XXXCURS_ in mBASIC. But you don't need to. You have a separate system of cursor control which works modulo 80, not modulo 256. And that's the only difference. In mBASIC, the screen values start at 1 (Home position), and run to 2000 (bottom right). Here are the algorithms for converting cursor value (x) to row and column: row=ip((x-1)/80)+1; column=mod((x-1),80)+1. Nothing much has changed except the value of the range (256 in all but mBASIC, 80 in mBASIC). Now for the examples.

In mFORTRAN, we had to pause screen output at any time (where's the cursor?) and print a message at HOME, and, after interrupt, return the cursor precisely where it was when the interrupt occurred--so we could continue to print at the exact point we left off. TGETCURS_ and TPUTCURS_ respectively memorize cursor position at interrupt and put the cursor back where it was, after interrupt. They work in the same way in APL and PASCAL.

A FORTRAN DEMO: We use the hex addresses of TGETCURS_ and TPUTCURS_, as given in watlib.exp on the language disk, so we need not convert to decimal (or to their negative addresses). This routine, whenever interrupted by pressing 'q', always puts the cursor back where it should be to continue printing after an interrupt. If you run and modify this demo you'll see instantly how the two library routines work. We include, at the end, a conversion of the

```
*curdemo.fortran
integer replace, row, col
character message, dummy, c, D
D=char(10)
message = "A demo: return cursor to old position."
print, char(12), "Enter 'q' to print message at HOME."
print, D, "Press RETURN to continue printing.", D,D
do n = 1,10
  print, char(11), message
  read, dummy
  if dummy = 'q'
```

```
    replace = sys(cnvh2i('b084')) [*TGETCURS_ line]
    print, char(1),char(6),'Message at Home'
    print, D, char(6), "Press RETURN to continue."
    read, c
    y = sys(cnvh2i('b087'), replace) [*TPUTCURS_ line]
  endif
enddo
row = replace/256
col = mod(replace,256)
print, D, char(6), "Last value of TGETCURS_ is: ", replace
print, "Row is",row,". Column is",col
end
```

last value assigned to TGETCURS_, to get both row and column, as outlined above.

Remove the comments with [* before you enter the program.

The method shown above is quite simple and works in all languages but mBASIC, in which cursor control is implemented in the interpreter. There, it is easy to write a simple little loop which you can interrupt at any time by OFF (ord 255); then memorize cursor position with the little trick Terry Peterson taught us, z = cursor(0), which assigns the present value of cursor position to 'z'. Next, you do whatever's needed during the interrupt. When that's finished, you slam cursor right back where it was with z = cursor(z). It's far simpler than using TGETCUR_ et. al., but in languages other than mBASIC you do not have this sweet capability. A word of warning: the method works a bit differently in immediate mode than in program, when cursor returns exactly where it was; in immediate mode, the position reported by z=cursor(0) is that where cursor comes to rest after the command is issued and you hit <RETURN>--always the next row.

```
100  loop
110    ... ! Printing away....
120    ... ! 'x' is the OFF key
130    get x ! does user want a coke?
140    if x = 255  ! She does...
140       z = cursor(0) ! Get cursor
150       ! position. Handle the in-
160       ! terrupt anywhere on screen.
170       ! Then quit the routine with
180       z=cursor(z) ! put cursor
190    endif          ! back...
200    ... ! resume printout
210  endloop
```

Here are more examples. First, we put the cursor in prison (for whatever reason) in a cell which is 30 columns wide and 5 rows deep. Try (just try!) to print anything outside the cell. Since we've excluded TAB (chr$(9)), you cannot escape. The cursor goes dead at the cell walls until you press CURSOR LEFT or CURSOR UP.

You can limit the user to a cell anywhere on screen in any language other than mBASIC by using TGETCURS_ and TPUTCURS_.

```
100 ! 'setlim.bd' sets screen limits
105
110 loop
115   get char : if char=0 or char=9 then 115
120   print chr$(char);
125   x=cursor(0)
130   row=ip((x-1)/80)+1 : col=mod((x-1),80)+1
135   while row>=5 or col>=30
145     get move
150   until move=11 or move=8 ! 11 is cursor
155 endloop           ! up, 8 is cursor left.
```

Some readers have complained that SuperPET does not allow the split screens of BASIC 4.0. Well, shucks, you can define your own split screens if you use the technique above, and can limit the user to whatever area you want whilst you print from program to the rest of the screen. Lest you think the code to do this must be long, we show at left four simple lines which irrevocably set right margin on the screen, and which use our

```
L$=chr$(8) or cursor left
```

```
710 x=cursor(0) : margin=40
720 while mod((x-1),80)+1 => margin
730   print L$; : x=cursor(0)
740 endloop
```

mod(x,80) directly instead of converting to column. We cannot drive the cursor past the margin set by 'margin', though we can pass it with the TAB key, only to be thrown out. Last, the strange beast shown here: z=cursor(mod(x+(80-x+1),80)) does quite well as a substitute for a carriage return, bringing the cursor to left margin of the screen from wherever it may be without forcing you to TAB or to use a CR. In fact, it will wrap the cursor to the next line, sans a CR or any tabbing. [In the example above, left, substitute a call to TGETCURS_ (with appropriate modulus) for x=cursor(0) in all languages but mBASIC.]

```
row 18 * 256 = 4608
col  1       =   +1
call value   = 4609
```

At left is a program to use TPUTCURS_ in mPASCAL. We want the cursor put on row 18, column 1. The cursor value for the call is computed at left, above. We did it manually, but you could easily write a procedure to compute it for you, based on row and column. Absolute cursor control in all the languages can be had at any time, together with firm screen control. And, sure enough, you can apply the method in assembly language as well. Suppose we want to read and print to the screen a directory, in two columns. We can't use the screen tabs (they may be set anywhere by the user). We can't just space over because directory strings are not the same length. But, it so happens the column value is returned by the function TGETCURS_ as the low byte in the D register, (which is the 8-bit B register). So, ignore the high byte row value in A register, and stuff in B the actual column value you want (see the LDB #40, above, left). Then pass the new value to TPUTCURS_ (all the values are right there, in the D register), and--voila! instant tab to screen column 40. Absolute screen control we have, in all languages, with ease and elegance. Quod Erat Demonstratum.

```
var
   putcurs:integer;

begin
  page;
  putcurs := 45191;
  sysproc (putcurs,4609);
  writeln('Here is where we put cursor');
end.
```

```
jsr tgetcurs_
ldb #40
jsr tputcurs_
```

◇◇◇

A BUG IN microPASCAL FILENAME LENGTH

by Robert I. Davis, Associate Editor
100 Darrow Drive, Pennington, N.J 08534

[Ed. See Gary Ratliff's Article, the Anatomy of MicroBASIC, this issue, for a clue on variable handing in SPET.]

Waterloo microPASCAL allows two alternative variations of a more general form of the standard PASCAL functions RESET and REWRITE (refer to Section G.6, File I/O Considerations, on p.131 of the mPASCAL Manual). For non-PASCALiers, RESET opens an existing file to be read from, while REWRITE opens a file to be written to and establishes a new file if necessary. There appears to be no problem with the first variation presented in the manual, but there is, unfortunately, a bug in the second variation which can cause very much confusion:

A filename defined as a variable of the type 'packed array of char' must have an EVEN total number of characters! In all of the trials that I've made, an extra character is added to the filename when you use RESET or REWRITE with a filename defined as a variable of the type 'packed array of char' with an ODD total number of characters. The extra character can be different each time; it seems to depend on the immediately preceeding machine operations. I ran across this bug when writing utility programs and procedures to create files which used filenames entered at the keyboard.

If you wish to investigate this bug as outlined, I suggest you use a scratch disk since you may encounter filenames containing characters which are very hard to duplicate. This will, in turn, make the files difficult to use or to scratch. To illustrate the bug, place a disk in drive 0 and then enter and run the program below. Afterward, the disk directory will be (with * being any character):

```
program tstopenfile1 ( output );

const
  filenamelength = 17;

var
  filename : packed array[1..filenamelength] of char;
  testfile : text;

begin
   filename := 'disk/0.odd_length';
   rewrite ( testfile, filename )
end.
```

```
0 "scratch disk    " zz 2c
1 "tstopenfile1.pas"  SEQ
1 "odd_length*"       SEQ
```

Note that a file "odd_length" has been opened, and an extraneous asterisk has been added! Next, change filenamelength to 18 and revise filename to even_length, and again run the program. A directory will then show:

```
1   "even_length"     SEQ
```

File 'even_length' is created without any extraneous character. Next, get back the original program, change REWRITE to RESET, and run. Note in the error message that the 'h' from the end of 'even_length' has been picked up and added to 'odd_length'. Enter: e writeln (filename) in debugger mode, and filename will be shown to be correct, as disk/0.odd_length, without the extra character. (You won't get the error message and go into debugger mode if the original extraneous character was h!) As a final check on this variation, change 'filenamelength' to 18, change 'odd_length' to 'even_length', leave RESET as is, and run. This time, the proper file, named 'even_length', is opened for reading.

As noted, there appears to be no problem with the first alternative variation. If you enter and run the following program, the whole disk directory will be:

```
program tstopenfile2;

var
  testfile : text;

begin
  rewrite ( testfile, 'disk/0.odd_length' )
end.
```

```
0  "scratch disk    " zz 2c
1  "tstopenfile1.pas"  SEQ
1  "odd_length*"       SEQ
1  "even_length"       SEQ
1  "tstopenfile2.pas"  SEQ
1  "odd_length"        SEQ
```

This time the file 'odd_length', which we originally tried to establish, has been opened without the addition of any extraneous character. Change REWRITE to RESET and run. File 'odd_length' opens for reading without any error.

I hope that this odd-even warning will save others from the confusion and frustration which I encountered.

◇◇

IRQ ROUTINES IN THE SWITCHED BANKS A few months ago, Terry Peterson and ye ed noticed that Bank 15 is used only with APL and COBOL, and so decided to stuff utility routines into Bank 15 to save on user memory. It worked fine until Terry tried it on 'nscroll', his advanced screen

dump to printer or disk (which also stops screen scrolling on demand). The program worked fine until you tried to load a language/facility, whereupon SPET crashed. It wasn't easy to find out why, or to get any interrupt routine in the banks to work reliably. Terry Peterson finally determined both the problem and the solution. The problem is in the operating system, which neglects to handle interrupts in the banks. Terry comments the op system code below, to show where the problem starts, and then provides the solution:

The Problem: Commented operating system code, as found in the system ROMs--

```
---------------------------------------------------------------------------
          ;Present 'banksw' ($BBF1) code to perform bank-switched call.
bankswi   PULS  Y,X       ; Pop two RTS addresses from the stack.
          PSHS  D         ; Save D accumulator.
          LDB   ,X+       ; Get bank number (and point to in-bank address)
          LDX   ,X        ; Get address
          CMPB  $0220     ; Same as current bank?
          IF    NE        ; No, so
             LDA   $0220          ;Save current bank number, stored in $0220
             PSHU  Y,A            ;on user stack.
*****        STB   $0220          ;Mark switch and store in bank pointer.
*****        STB   $effc          ;Then latch and make the switch.
             PULS  D              ;Restore D accumulator.
             JSR   ,X             ;Go do your subroutine.
             PSHS  D              ;Save D again (only).
             PULU  Y,B            ;Get back orig. bank & caller's address.
*****        STB   $0220          ;Put it back in $0220 for reference, and
*****        STB   $effc          ;latch and switch.
             LDD   ,S++           ;Restore D (& set Zero flag) & pop SP back
             JMP   ,Y             ;Back to caller
          ENDIF
          PULS  D         ; No need to switch banks.
          PSHS  Y         ; Restore RTS address.
          JMP   ,X        ; Go to subroutine.
---------------------------------------------------------------------------
            ; Present 'bankinit' code:
bankinit  LDU   #$02ff    ; Start user stack pointer at $02ff
          LDB   #$00      ; & with bank #0
*****     STB   $0220     ; Put the pointer in.
*****     STB   $effc     ; and do it.
          RTS
---------------------------------------------------------------------------
```

***** These lines of code MUST be replaced by the following in order to allow IRQ routines to reside in SPET bank-switched memory! Any other references to $0220 and $EFFC must likewise be so replaced.

```
          bsr     safe_bank_switch ; Substitute this line for ***** lines, above,
                                   ; which calls the subroutine below:
```

The following subroutine is employed in order to avoid the disaster that occurs when an IRQ-generated call to a routine in bank-switched memory happens between the first and second lines marked '*****', above, when there are other routines in the same bank as the IRQ routine.

```
safe_bank_switch        equ     *

        sei             ;SET INTERRUPT MASK to disable regular interrupts!
        stb     $220    ;NOW, change pointer
        stb     $EFFC   ;and set bank-select latch (for sure).
        cli             ;and THEN clear the interrupt mask....
        rts
```

Since the operating system bug is embalmed in ROM hardware, you must work around it. Terry wrote two programs, 'hello' and 'gscroll', which patch the bug. If you load 'hello' from menu first thing, it gives you a chance to set time and date, which you can bypass, and then loads 'gscroll,' a multi-purpose dump which resides in banks 15 and 14, out of user memory. With 'gscroll' you can keep the screen from scrolling at any time by pressing SHIFT OFF, and resume it with the same key, whilst you dump screen to printer with SHIFT OFF TAB, or to disk with SHIFT OFF LEFT-ARROW. The program obligingly either opens a file called 'screen dump', or appends your data to that file if the file already exists. The R/W switch must stay in PRG position throughout your session. You can load any language/facility in SuperPET except COBOL or APL under this arrangement. Terry has also revised the keypad for some instant keywords--such as PF3, which gives you p ieee4 in one stroke--and you have the alternative of going back to the standard keypad arrangement with PF1. Both 'hello' and 'gscroll' are on the ISPUG Utility disk defined in this issue, with full source code.

◇◇

TO KILL A MOCKINGBIRD... Last issue, we showed two simple ways to load a machine language module for use in the languages, but said nothing about loading one <u>from</u> language because every attempt resulted in loud cackles from the mockingbird. Though the other DOS commands work in program, the old "L0:filename" routine won't, and we couldn't find any other way to load an ML module from program at the right address. So we asked Terry Peterson for help. Back came a routine which loads a module from language. It must be the first part of a program (and don't define ANY variables before you RUN it!). Before you read further, take a look at the loader below.

We sent an early version of the loader to Loch Rose, and he pointed out that in a cold start with mBASIC, a giant a$ [rpt$('*',9000)], generated either in immediate mode or in program, would overwrite the ML module just loaded. Obviously, there's a pointer to top-of-strings in microBASIC which is reset at a CLEAR or on a call to the mED, but which is <u>not</u> automatically reset by a mere poke of memend_ itself, down at $22. After you poke a new memend_, you must <u>somehow</u> reset that darned top-of-strings pointer.... Well, we did it with a dump of the keyboard buffer, an automatic call to EDIT, and an automatic RUN therefter, with thanks to Frank Brewster (Vol. I, p. 122 ff). His trick indeed resets the top-of-string pointer (any call to mED resets all pointers and closes all files). (Warning on the first ten copies of the ISPUG utility disk: file ml.loader:bp must be modified to conform to the example below.)

Second secret: 'chain' the machine-language module. SuperPET then thinks that a new mBASIC program has been loaded which overlays and wipes out the old. So it begins execution at the lowest line number in what it thinks is the new program. But there isn't any 'new' program--SPET just re-executes the old one. On the first pass, before chaining, variable 'loaded' is 0; the ML module CHAINs in. The value of 'loaded' (now 1) is passed to the 'new' program by the command to chain 'names' at line 130. On the next pass, SuperPET executes the 'else' clause, resets memend_, DELETEs the loader lines, calls EDIT to reset the darn

pointers, and gives itself a RUN which starts at line 200. The entire process is Rube Goldberg at his finest, but it works--automatically.

We hope the PASCAL and FORTRAN folks pick up on this and find a way to load from those languages. APL maybe will be less of a problem. Send in your solutions!

```
100 if loaded=0  ! ml.loader:bp
110    loaded=1
120    poke (hex('22')),hex('7f'),hex('ff')  ! Set memend_ to $7fff
130    chain "alphasys:6000/6,prg",names     ! Load the ML module.
140 else
150    poke hex('22'),hex('60'), 00          ! Reset memend_ to $6000
160    open #40, "keyboard", output
170    print #40,"delete 100-190"+chr$(13)+"edit"+chr$(13)+"run"+chr$(13)
180 endif
190 stop                 ! STOP dumps the keyboard buffer, which sends commands
200 a$=rpt$('*',9000) ! to delete lines 100-190, to call EDIT (reset pointers)
210 print a$(1:5)      ! and then to RUN from the first remaining line, 200.
220 sys hex('6006')    ! SYS any time.      ! This ain't elegant, but it works.
```

This program resets memend_ (found at $22 and $23) to $7fff on all runs. Without the reset, you'll crash if you again try to load the ML module or if you've set memend_ lower than $7fff in a previous program. Suggest you again reset memend_ to $7fff by a POKE at the end of program, so you get back all of user memory.

Note the filename for the ML module: alphasys:6000/6. The first number is the 'memend_' set, the second is the 'SYS' address. With such filenames on directory you have all data needed to write a program with the right ML module addresses. The reason for the two different addresses should be clear if you read the first few lines of alphasys:6000/6, below:

Program:			Addresses:	Comment:
start	ldd	#start	$6000	This section is designed to set memend_
	std	memend_	$6003	if loaded from the monitor. Since it
	swi		$6005	never gets run if loaded from language, memend_ must be set from your program.
	stu	ustack	$6006	The language module starts here.

AS SIMPLE AS INVENTING THE WHEEL We hereby award James Sweeny of 15A Mohonk Ave., New Paltz, N.Y. 12561, ISPUG's Pat Pending Cup (in memory of that inventive genius). Follow this: "I found a way to set up my printer from the mED. You create a disk file for each command; the command contains only one line--the character sequence to be sent. Any time you want to change the printer, just use the COPY command from the mED. For example, on the PROWRITER the command for proportional spacing is ESCAPE P, or chr$(27); chr$(80). My file "P proportion" contains this sequence. To use it, I type: copy "P proportion" to ieee4, and my printer is set. Anything in the mED is left un-unchanged." So we tried Jim's method on our printer, and, by golly, we can make the darned thing dance the hornpipe--from the mED. Find below a couple of sample programs which create printer command files on disk. Are the rest of you ISPUGgers as thickheaded as ye ed in not figuring this out long since?

```
10 open #3, 'P proportion', output      ! File holds ESCAPE P to set PROWRITER
20 print #3, chr$(27); chr$(80)         ! to proportional spacing mode.
30 close #3                             ! From Jim Sweeny
```

```
10 open #3, 'setmargin', output                ! File  sends  56  spaces,  ESC  9
20 print #3, rpt$(' ',56);chr$(27);chr$(57) ! to set DIABLO left margin
30 close #3                                     ! at  56.  Works  well  for  ye  ed.
```

◇◇

A SUPERPET FILENAME CONVENTION Few of us are patient enough to load and try each of the sixteen languages or facilities SuperPET can use when a disk arrives with unidentified files. Where does a PRG file run? Where should a SEQ file be read or used? Without a filename convention, the disk might as well be written in Sanskrit.

MicroPIP, Waterloo's beautiful program for file-handling (Vol. I, No. 9), gave us a way to sort, copy, and rename files by function, language, and filetype--but only if files are named in a systematic way.

We've therefore put together the scheme below, which has been tested, modified, and retested for five months. We've coordinated it with TPUG; Bill Dutfield, who is the SuperPET disk librarian, suggested several excellent changes. From here on out, ISPUG and TPUG will use the system to identify files. All ISPUG disks of recent vintage use it, so you can tell in what language or facility every program runs, and the purpose or type of the file.

(1) We distinguish 6809 files by using the colon ':' or the '_' between the filename and the language/facility identifier, as in: filename:bd (which identifies a demo file in microBASIC (the colon won't work in mPASCAL). Files in 6502 are identified by a period as a suffix separator. If all filenames on a disk are in CAPITAL letters, the colon or '_' are the only way to tell which files run in 6809 and which in 6502.

(2) Language or Facility Suffixes:

6809				6502			
:b	microBASIC	:a	microAPL	.B	BASIC 4.0	.PC	Paperclip
:f	microFORTRAN	_p	microPASCAL	.WP	WordPro	.M	6502 ML
:c	microCOBOL	:e	microEDITOR	.WC	WordCraft	.C	COMAL
:fo	forth (6809)			.BC	Compiled 4.0	.FO	Forth (6502)
(We have FORTH coming, pun intended)				.R4	RTC 4	.SS	SuperScript

Development (Assembler) files are already distinctly identified in 6809, except for load modules from which '.mod' has been stripped. Such PRG modules can run either from main menu, or from the monitor. Since you never know which:

:'go' or load address designates such modules for the monitor, and
:men designates modules designed to be loaded from menu.

(3) Capitals and Lower Case: Preferably, 6809 files are in lower case and 6502 files in lower case in 6502 (which puts them in upper case in 6809). This leaves no doubt at all on which side of SuperPET they run. In 6502, you can read only those filenames which will run there--highly desirable unless you still can't handle the DOS in 6809 (shame!). This convention distinguishes 6809 and 6502 files even if all filenames are in capital letters.

(4) Type of File Designators: Useful 2nd and 3rd character suffixes, which identify the file by type, function, and status.

t	Text File	i	Index File	z	Not debugged
p	Run File (a program)	u	Utility File	l	Library file
ws	APL workspace	r	Reference File	d	Demonstration prg
fn	APL Function	x	Executive (driver)	s	Subroutine (slave)
in	Instruction file (it is an index file as well)			dd	Data File

(5) Examples, Various Languages: Filename:

APL A workspace, 'EXAMPLE'	example:aws [In APL, a paren will
APL A revised version of 'EXAMPLE'	example1:aws appear, not a colon.]
APL A text seq file 'STATS'	stats:at [SEQ filetype on Directory]
APL A utility to list to printer	list:au
APL Same utility, not debugged	list:auz
mBASIC Utility Program 'tabset'	tabset:bu
mBASIC driver program, 'jewel'; loads and runs sub-programs/routines	jewel:bx
1st routine used by jewel:bx	jewel1:bs or jewel.1:bs
mBASIC library procedure, used in many programs to print to printer	printit:bl
mFORTRAN program, buggy, 'calc_yield'	calc_yield:fpz
mFORTRAN program to run, 'calc_yield'	calc_yield:fp
The first modification to the above	calc_yield1:fp or calc_yield.1:fp
The second modification to the above	calc_yield2:fp or calc_yield.2:fp
mCOBOL utility, sorts lists	sortlist:cu
microEDITOR text file, instructions	instruct:e or describe:ein
Instructions for COMAL, in COMAL	DESCRIBE.CIN
Instructions for APL, in APL	describe:ain
UDUMP load module, menu	u:men
UDUMP load module, monitor	u:7e60

Note that we use the 'go' address of monitor files to designate such files. You must know that address to give the 'go', and putting it on directory is a simple way to find and remember it--as well as a signal that it does not run from main menu. Last, the 'z' identifier clearly tells you that you have a buggy program which isn't safe to use. If you have several program versions, it's handy.

If you send material to TPUG or to the Gazette, please use this scheme. We'll return the favor on all disks issued for SuperPET from ISPUG and TPUG.

(C) 1984 ◊◊◊ *THE APL EXCHANGE* ◊◊◊ *STEVE ZELLER*

In Vol.I, p. 136, Dick Barnes reviewed a new terminal emulator called COM-MASTER (CM). Since then, I have been working with its author, Dan Jeffers, on the APL side of CM. The result is very rewarding. Of particular interest to APL users is the representation (in either direction) of A-Z "underbar" characters as A-Z

"reversed". This feature is extremely useful (I can now read all variable and function names on IP Sharp). This feature alone sets out CM as the best APL terfinal emulator available. But there's more!

CM employs a command language to set its various parameters and these commands can be collected in a text file and edited with Waterloo's editor. This allows movement between mainframes simply by loading a new setup file while in CM's main menu. The shifted numeric pad functions as a set of ten program function keys (PF.,PF0,...,PF9) and these can be assigned character strings. Their use in APL is a little tricky, however, since the right paren, ')', in APL is a quote '"' in ASCII, and CM uses the quote as a delimiter in command strings. Thus, setting up PF7 to be the APL string ')VARS' requires the command: PF7= """vars". ASCII control characters may also be included in the string. This capability has numerous possibilities. During the initial setup, for example, the PF keys can be configured to send autodial messages to the modem and to send the logon sequence to the host. Next, another command file can be read in to reconfigure the PF keys for APL programming and system commands. If you must continually edit a function on the host, you can set up a PF key to open the function for editing and then finish the line with the line number you wish to edit bebefore you hit <CR>. It saves an enormous amount of time.

The upload/download capabilities are well thought out. Filenames are shown on screen at all times and in "reverse" when active. The <LF> characters can be stripped out of files during downloading so that they conform to the Commodore file convention. When uploading files to the host, CM will either: (1) send a stream of characters continuously, (2) pause for a specified time interval before sending the next record, or (3) wait for the receipt of a specified character. This last feature, alone, justifies having CM. An APL system sending a <BELL> as a prompt is ideal for this feature. Files can be uploaded quickly, with little danger of losing characters.

The result is an excellent terminal package with APL features that are very useful and not available in any other package. The terminal emulator is available from Quality Data Services, 2847 Waialae Avenue, #104, Honolulu, Hawaii, USA 96826 for $95 (US). [Ed. This package also is a general-purpose terminal emulator useful for all SuperPET telecommunications.]

In this issue, I examine the several approaches to generating menus in APL. Menus are very useful in applications used by those who lack experience in APL or in programs used infrequently. A menu essentially embeds instructions in a program itself and allows that program to control entry at crucial stages in its execution. While menus are widely believed to be "user friendly," I find that too many menus as actually written have the opposite effect.

The natural way to represent a menu in APL is by a character matrix with the rows of the matrix corresponding to the choices available. You can use the function below to construct matrices for this purpose. All the examples shown here assume a menu that fits nicely on one screen.

```
      MENU1←BUILD_MENU
ENTER: NO. OF ROWS AND COLS IN MENU MATRIX
□:
      5 10    |-----------------------------------------------------
ENTER ROW 1   |
```

```
CHOICE 1      |        ∇BUILD_MENU[□]∇
ENTER ROW 2   | [  0]    R ← BUILD_MENU ;N;I
CHOICE 2      | [  1]   ⍝BUILDS MATRIX FOR USE IN MENU ROUTINES
ENTER ROW 3   | [  2]   S1:'ENTER: NO. OF ROWS AND COLS IN MENU MATRIX'
CHOICE 3      | [  3]   →(2≠ρN←□)/S1
ENTER ROW 4   | [  4]   R←Nρ' ',⍳I←0
CHOICE 4      | [  5]   S2:REVERSE 'ENTER ROW ',⍕I←I+1
ENTER ROW 5   | [  6]     R[I;]←N[2]↑⍕
CHOICE 5      | [  7]     →(N[1]>I)/S2
DONE          | [  8]   'DONE'
```

The first method of presenting a menu is METHOD1, shown below. I simply clear the screen, add numbers to the menu matrix and wait for a response. The selection is edited by GET_ANSWER, which checks to see if the response is numeric and if it falls within the prescribed range. METHOD1 then returns the choice to the calling program.

```
      ∇METHOD1[□]∇
[  0]    R ← TITLE METHOD1 MENU ;N;ANS
[  1]   ⍝THIS IS THE FIRST METHOD OF MENU CONTROL
[  2]   ΔTCFF,(CENTER REVERSE TITLE),ΔTCNL
[  3]   N←(1↑ρMENU)+R←0
[  4]   (4 0⍕(N,1)ρ⍳N),'-',MENU
[  5]   ΔTCNL,REVERSE 'ENTER CHOICE (OR <CR> TO QUIT'
[  6]   R←GET_ANSWER N

      ∇GET_ANSWER[□]∇
[  0]    R ← GET_ANSWER N ;ANS
[  1]   ⍝GETS RESPONSE TO MENU
[  2]   S1:→(0=ρANS←⍞)/R←0
[  3]   →(∧/ANS∈'0123456789')/OK
[  4]     (2ρ□TC[□IO+3]),REVERSE ΔMSGLENGTH↑'BAD ENTRY, TRY AGAIN'
[  5]     →S1
[  6]   OK:→((R←⍎ANS)∈⍳N)/0
[  7]     (2ρ□TC[□IO+3]),ΔMSGLENGTH↑REVERSE 'OUT OF RANGE, TRY AGAIN'
[  8]     →S1

      ∇REVERSE[□]∇
[  0]    R ← REVERSE S
[  1]   R←□AV[128+□AV⍳S]

      ∇CENTER[□]∇
[  0]    R ← CENTER MSG
[  1]   ⍝CENTERS TEXT STRING ON SCREEN
[  2]   R←79↑(((⌊(79-ρMSG)÷2)ρ' '),MSG

      ∇UNREVERSE[□]∇
[  0]    R ← UNREVERSE S
[  1]   R←□AV[¯128+□AV⍳S]
```

```
|----------------------
|  ΔMSGLENGTH←25
|  ΔTCNL←□TC[□IO+6]
|  ΔTCFF←□TC[□IO+4]
|----------------------
```

```
|------------------------------
| THESE ROUTINES ARE USEFUL WHEN
| DISPLAYING MENU CHOICES AND
| TITLES. NO CHECKING IS DONE,
| HOWEVER. HENCE, A STRING TO BE
| "REVERSED" MUST BE IN THE LOWER
| HALF OF □AV WHILE A STRING TO BE
| "UNREVERSED" MUST BE IN THE UPPER
| HALF. SIMILARLY, THE STRING "TITLE"
| IS ASSUMED TO BE LESS THAN 79 CHARS
|-----------------------------------
```

The second approach allows the user to verify the selection. METHOD2 treats the screen as a relative file. After the user enters a selection, the relevant record is rewritten with the choice reversed. A message at the bottom of the screen

then asks for confirmation. Display is somewhat slower with this approach because of the looping, but it provides a way to ensure correct responses.

```
      ∇METHOD2[□]∇
[  0]    R ← TITLE METHOD2 MENU ;N;I
[  1]   ⍝THIS IS THE SECOND METHOD OF MENU CONTROL
[  2]   'TERMINAL' □CREATE 1
[  3]   ΔTCFF,⍳R←0
[  4]   □SEEK 1,0
[  5]   (CENTER REVERSE TITLE) □PUT 1
[  6]   N←(1↑⍴MENU)+I←0
[  7]   S1:□SEEK 1,2+I←I+1
[  8]     ((4 0▼I),'-',MENU[I;]) □PUT 1
[  9]     →(N>I)/S1
[ 10]   S2:□SEEK 1,22
[ 11]   (REVERSE ΔMSGLENGTH↑'ENTER CHOICE') □PUT 1
[ 12]     →(0=R←GET_ANSWER N)/END
[ 13]   EXIT:□SEEK 1,2+R
[ 14]     ((4 0▼R),'-',REVERSE MENU[R;]) □PUT 1
[ 15]   S3:□SEEK 1,22
[ 16]     (REVERSE ΔMSGLENGTH↑'OK? (Y/N)') □PUT 1
[ 17]     →('Y'=1↑□)/END
[ 18]     →S1
[ 19]   END:□UNTIE 1
```

The final technique presents the menu as a horizontal bar (Note: you need to be sure that the menu will fit on one line). This has the obvious advantage of not using up the entire screen. The first row of the menu is the default choice, and a quick <CR> selects it. Other selections are also shown in reverse. The use of <CR>, without typing a choice, is different from the previous two examples. In both METHOD1 and METHOD2, just a <CR> results in zero being returned to the main calling program. This allows one to "back out" of a menu. Here, <CR> signifies choice. Consequently, an option to "exit" or "quit" may be needed on the menu bar.

```
      ∇METHOD3[□]∇
[  0]    R ← LINE METHOD3 MENU ;N;I;SPACE;BAR
[  1]   ⍝THIRD METHOD OF MENU DISPLAY
[  2]   N←⍴MENU
[  3]   SPACE←0|⌊(79-×/N)÷N[1]
[  4]   BAR←MENU,(N[1],SPACE)⍴' '
[  5]   'TERMINAL' □CREATE R←1
[  6]   S1:BAR[I;]←REVERSE BAR[I←R;]
[  7]   □SEEK 1,LINE
[  8]   (,BAR) □PUT 1
[  9]     BAR[I;]←UNREVERSE BAR[I;]
[ 10]   REVERSE ΔMSGLENGTH↑'ENTER CHOICE'
[ 11]     →(0≠R←GET_ANSWER N[1])/S1
[ 12]   EXIT:□UNTIE 1
[ 13]   R←I
```

Armed with an imposing array of menus, are we ready to compete with the visual interface in Apple's Macintosh? The "Mac" goes well beyond most other micros by providing very fast, "pull down" menus and mouse-driven selection. I think this is the way to go but we'll never be able to pursue it on the SPET.

Dick Werner (Elkhart, Indiana) contributed an APL function to convert a measure in inches to one in feet and inches (it's a lot easier in centimeters!).

```
      ∇FTIN[⎕]∇
[  0]    R ← FTIN X ;T1;T2
[  1]   ⍝CONVERTS X REAL INCHES TO α, FEET AND INCHES (BY MIKE WERNER)
[  2]   T1←ρX
[  3]   T2←(T1,2)ρ⍉ 0 12 ⊤X
[  4]   T2←(T1,4)ρ((2×T1),2)↑(⍕((2×T1),1)ρT2) ⍝CRUNCHES AND STRIPS THE BLANKS
[  5]   T2←1 1 0 1 1 0 0\T2                  |-----------------------------
[  6]   T2[;3]←T2[;6]←T2[;7]←''''            | X IS A SAMPLE VECTOR WITH TWO OR
[  7]   R←T2                                 | ELEMENTS, NONE OF WHICH CAN EXCEED
       ∇FT_TO_IN[⎕]∇                         | 1199.
[  0]    R ← FT_TO_IN X ;N                   |---------------------------------
[  1]   R←(N,10)↑((N←ρ,X),8)ρ4 0⍕⍉0 12⊤X     | HERE IS MY VERSION OF MIKE'S FN.
[  2]   R[;5 9 10]←⎕AV[⎕IO+75]               |---------------------------------
EXAMPLE:
      FTIN 54 98 156          |              FT_TO_IN 54 98 156
 4' 6''                       |           4'  6''
 8' 2''                       |           8'  2''
13' 0''                       |          13'  0''
```

Finally, there are two books that are full of interesting APL functions. The first is by Francis Anscombe, a professor in statistics at Yale, entitled: "Computing in Statistical Science through APL" (SprInger-Verlag, 1981). The Appendix presents the statistical package used throughout the book. A disk containing these functions was recently contributed to TPUG (available as TPUG T7 in the SuperPET series). TPUG sent it to ISPUG. You can get the disk by sending $10 U.S. to the Editor, PO Box 411, Hatteras, N.C. 27943 in 8050 format, or to the Secretary, 4782 Boston Post Road, Pelham, N.Y. 10803 in 4040. I highly recommend the book (around $25) and disk. [Ed. Order the "Anscombe" disk.]

Another academic, Prof. Ulf Grenander of Brown University, has written "Mathematical Experiments on the Computer" (Academic Press, 1982). The title notwithstanding, it is full of APL functions to do algebra, analysis, arithmetic, asymptotics, geometry, graphs, probability and statistics. This volume is somewhat more expensive than Anscombe's (around $40) but is definitely worth looking at.

∩∩∩

6425 31*ST ST., N.W., WASHINGTON, D.C.* 20015 *U.S.A.*

∪∪∪

ANATOMY OF MICROBASIC Part 1

by Gary L. Ratliff, Sr.

This article introduces you to the memory management methods of Waterloo microBASIC and contrasts them with the well-documented methods of Commodore BASIC. Some details are similar; many are different. To save on space, I will identify Commodore BASIC as cBASIC, and microBASIC as mBASIC.

Part A: Simple Variables.

Both languages allow long variable names in program, but a long name is truncated by cBASIC to the first two letters; cBASIC further doesn't allow keywords in a variable name (FORTRAN is disallowed, for example) while this practice is not at all discouraged in mBASIC (fortran is perfectly acceptable). Why? A variable name is never entered within the tokenized line of mBASIC! In contrast, cBASIC has the variable name within the line, and its 'crunch token' routine will find any 'for' or 'to' you might include in a variable name and will convert it to a token. Let's contrast the two methods:

10 a=1 cBASIC will convert the line at left 1) into the ASCII code for 'a', 2) the token for '=', and 3) the ASCII code for '1'. mBASIC instead 1) translates the variable name 'a' into a pointer to the variable storage area, 2) converts '=' to a token, and 3) if the value (here '1') is small, makes an integer of it; if it is large, converts the value into a token for a string with a given length, in the form of ASCII codes for the string.

Waterloo avoids conflict between keywords and variable names by never having the variable name appear within the tokenized line! 'a' could as well be 'absol' or even 'absolute_value_of_variable_one' (up to 31 characters), and as the first variable named in the program, would be seen as 00 01 (a pointer to the first variable in the variable storage area). You may observe this by using the monitor from mBASIC. The command: SYS 61631 will take you into the monitor; don't enter the monitor from the microEDITOR in mBASIC, for this converts the program from its tokenized form into a different form for the mED! The two forms are not the same.

By experiment we've discovered that the length of a simple variable name is always odd, as we'll show with the examples below. As you follow them, remember that the ASCII codes are in hex ('a'=$61, 'b'=$62, 'c'=$63, and 'd'=$64). The first byte in the assignment of each line is the length of the variable name. Thus, line 10 of Program 1 becomes: 01 61 (one byte, variable name is 'a'). The full assignment for the first 3 lines of program 1 follows, with xx indicating space for values. Below is what you see in the variable storage area (The values of the variables are not set down, because they are in excess 128 notation, and I do not want to obscure this article with that! Instead, I show the values as: xx xx xx, etc. The excess 128 notation is used only for floating-point values):

Program 1	Program 2
10 a=1	10 aa=1
20 b=2	20 bb=2
30 c=3	30 cc=3
40 d=3	40 dd=4

Example 1

Byte No.(hex):	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15
Value:	01	61	xx	xx	xx	xx	xx	01	62	xx	xx	xx	xx	xx	01	63	xx	xx	xx	xx	xx
Variable Pointer:	00 01 (to variable a)									00 08 (to variable b)						00 0f (to variable c)					

I also show the values in the variable pointer table. You can see how the location of each variable is stored (note the byte number on line 1). Now, let us see what happens with program 2 (remember $20 is ASCII 32 decimal--a space):

Example 2

Byte No.(hex)	1	2	3	4	5	6	7	8	9	a	b	c	c	e	f	10	11	12	13	14	15	16
Value:	03	20	61	61	xx	xx	xx	xx	xx	03	20	62	62	xx	xx	xx	xx	xx	03	20	63	63
		--									--									--		
Variable Pointer:	00 01 (to variable aa)									00 0a (to variable bb)									00 13 (to var. cc)			

Note how a variable name of even length has a space prefixed (see the '--' under each $20 in the 'Value' line, above--to always make the length of the variable an odd number. Why the insistence on an odd number? Let us come back to this question later. Meanwhile, note that any floating point variable value creates the following storage requirements: 5 bytes for the numerical value (note the vacant five bytes in the tables above), plus one byte for the length pointer or token at the start, plus one byte for each character in the variable name, plus an optional byte for the odd-making space.

Now, if we change the program from floating-point values (default) to integer values, see what happens (convert Program 1, above, to read a%=1, etc.):

Example 3

```
Byte No.(hex)   1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10
Value:         41 61 00 01 41 62 00 02 41 63 00 03 41 64 00 04
------------
Variable       00 01       00 05       00 09       00 0d
Pointer:       (to a%)     (to b%)     (to c%)     (to d%)
```

[Note: Here we use positive values, and they appear as such. If we used negative integers, however, they would appear as two's complement. $7fff is the highest integer you may use (decimal 32767); $8000 is the lowest negative in two's complement form, at -32768 decimal.]

You might think $41 ('A') identifies integers--but wait till later. The ASCII code for the variable name follows, suffixed and terminated by 00, and then we find the value of the variable. We begin to understand why programs using integers in mBASIC are so much faster than those using floating-point values. But before we draw any firm conclusions, let us look at string storage for the four strings following: a$="one", b$='two", c$="three", and d$="four". Here is what we find in the variable storage area:

Example 4

```
Byte No.(hex)  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10
Value:        21 61 7f xx 21 62 7f yy 21 63 7f zz 21 64 7f zx
----------
```

Comment: a$ found at 7fxx, b$ at 7fyy, c$ at 7fzz, d$ at 7fzx, where the letters after 7f stand for the actual low byte of the address.

Note the difference: the variable storage area does not contain the string, as the variable storage area does for integers and floating point values. Instead, it contains a pointer to high user memory (note the $7f xx, etc., above). And, indeed, if we look in high user memory at those addresses, we find the strings. (Note the warning in the last issue of the Gazette about setting end of user memory [memend_, at $22] before loading or using an mBASIC program when you wish to SYS from it to a machine-language module in high user memory.]

We can conclude as follows: the type of simple variables is encoded in mBASIC in one byte, whose format is: tttlllll. If the high three bits (ttt)=000, then the type is floating-point. If ttt=010 then the type is integer; and if ttt=001, the type is string. The format of lllll is xxxx1, as this value is always an odd number. These five lower bits represent the length of the variable name. We therefore can parse that first byte in all the examples above to see how it is generated and to make our anlysis clear:

Example Number:	Value of 1st byte (hex)	Composition of 1st byte (binary) ttt	lllll	Summed Value of 1st byte (hex)	Parsed Meaning Type of Variable:	Length of Var. Name:
1	01	000	00001		Float. Pt.	1 character
	OR		00001	(length of variable name)		

	Final Value:	000	00001	$01 (binary 0000 0001)		

```
2         03             000     00001                          Float. Pt.  2 characters
                    OR           00010 (length of variable name)
                                 -----
        Final Value:   000       00011         $03 (binary 0000 0011)
-----------------------------------------------------------------------------------------
3         41             010     00001                          Integer     1 character
                    OR           00001 (length of variable name)
                                 -----
        Final Value:   010       00001         $41 (binary 0100 0001)
-----------------------------------------------------------------------------------------
Ex.       43             010     00001                          Integer     2 characters
(Theory)            OR           00010 (length of variable name)
                                 -----
        Final Value:   010       00011         $43 (binary 0100 0011)
-----------------------------------------------------------------------------------------
4         21             001     00001                          String      1 character
                    OR           00001 (length of variable name)
                                 -----
        Final Value:   001       00001         $21 (binary 0010 0001)
```

With the table above, we see how that first byte is constructed, and the reason for the odd number in the variable storage area--the length of the variable name being OR'd with 00001, which inevitably causes an odd number. Because of this, Waterloo is forced to stuff in a space if the variable name is composed of an even number of characters. Next issue, we'll delve deeper into the construction of microBASIC.

◇◇

PETCOM 1.2 : A SOFTWARE REVIEW

by John D. Frost
Associate Editor, Telecommunications
7722 Fauntleroy Way, S.W.
Seattle, Washington 98136

Ph.D. Associates, Inc., Suite 200, Kinsman Bldg., 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3, kindly arranged a "loan for review" of its telecommunications software for SuperPET, which is called PETCOM 1.2--a professionally-done package for the 6809 side of SuperPET. It provides full upload/download capabilities for both SEQ and PRG (I can them binary) files, lets you use DOS commands while in the program, is menu-driven and very user-friendly, and gives you the bonus of an exceptionally well-written manual of 84 pages.

PETCOM is written for the user who wants to communicate with a variety of host computers (mainframes, minis, or micros) either with a modem or a direct connection. It emphasizes the larger scientific mainframes and higher baud rates, but is equally at home working with a local bulletin board, a school mainframe, another SuperPET, or with a commercial database like CompuServe. PETCOM can coreside with the mED and with three of the five Waterloo languages (mBASIC, mFORTRAN and mPASCAL); it supports a JSR (jump to subroutine) machine-language call.

Apparently the author of PETCOM has "done it all" and is eager to share his experience. The PETCOM manual is a tutorial on the communications process, and guides you in detail thru every aspect of a telecom connection, with good examples and good diagnostics when that first attempt doesn't quite ring true.

Significantly, you can use the various DOS commands while running PETCOM without having to load or use the mED. The program adds to the usual DOS offerings of Directory, Rename, Scratch, etc., a way to manipulate the COPY capability so you

can list portions of previously downloaded files to the screen or print files to either a Commodore or ASCII printer. You can copy part of a file, extract portions of large files and copy them into smaller files, single step a listing to the screen, and execute other useful COPY instructions.

PETCOM arrived configured to communicate with a DEC-10 computer, with defaults set to an 8-bit data word, a baud rate of 1200 and all CAPS from the keyboard. The manual clearly tells you how to reconfigure the program for your particular host and for your needs; you'll find that manual a fine tutorial on the reasons for such changes. You can't help but learn as you configure the program to the system of your choice with a few quick keystrokes in response to menu prompts. You need only reconfigure once, as PETCOM gives you a way to save a "user-optimised" version of the program to disk. You can save a whole family of such PETCOM programs, each tailored for a specific communication partner.

I was able to configure PETCOM to communicate and upload/download to a company mainframe and to an IBM PC, at both 300 and 1200 baud, using a Hayes Smartmodem. PETCOM easily supported a connection with CompuServe and with the local Commodore Bulletin board at 300 baud. But--the up/downloading protocols of a Punter (Commodore) bulletin board and the Vidtex protocols of CompuServe are not supported by PETCOM. I had difficulty transferring binary (PRG) files as I was unable to configure the host to receive binary format. My impression was that PETCOM had sucessfully retrieved the PRG file from SuperPET's disk drive and was transmitting properly but my host just wasn't receptive to the format.

Some noteworthy features of PETCOM set it apart from previous telecom packages. You select ASCII CONTROL characters one of two ways: 1) The primary method reassigns the SHIFT key as the CONTROL key, and sets the keyboard to a "caps locked" condition. You transmit any CONTROL character by holding down SHIFT while you press the selected character key, but you cannot enter lower-case characters from the keyboard; 2) In the secondary system ("caps unlocked") you're allowed both upper and lower case letters from the keyboard, but CONTROL characters are available only from the SHIFTED keypad. In this mode, the CONTROLs are limited to C,D,N,O,Q,S,T,U,V,Z and <BREAK>.

After you open a download file, you can start and stop the actual filing to disk by toggling the RVS key, and so literally pick and choose the material you commit to disk as you browse thru a long data base.

PETCOM lets you use a different set of handshaking parameters for uploading than you use for terminal mode. This permits you to specify different End of File (EOF) characters for the upload, to screen out embedded line feeds that really don't need to be transmitted, to control a time-out function for which you're willing to wait for a host response, and to specify SuperPET's response to the host prompt 'send next record' if one is available. These features are very desirable at high baud rates and when you transfer binary files.

Most usefully, you may send a one-line command to a host as the "first" line of an upload from a SuperPET disk file. You might, for example, command "create jdf.txt" at the start of the uploaded file, so you don't have to send the same message in terminal mode.

In the three weeks I had to familiarize myself with the package, I became quite attached to PETCOM's menus and general ease of operation, and very appreciative

of the accompanying instruction manual. The menu-driven selection screens are efficient, especially when you're experimenting with a variety of telecom hosts.

I was (and still am) bothered by the SHIFT key as a CONTROL key, since I am well-accustomed to the RVS key for that function in WordPro, in NEWTERM and in most other software packages. I am likewise spoiled by NEWTERM, which gives you direct access to, and the capability to change any or all of, the incoming/outgoing translate tables. If those tables were accessible in PETCOM, I'd certainly change them to get some CONTROL characters which are not now available (one of my databases calls for a CONTROL K, which I can't get in PETCOM from the shifted keypad). And I'd like to be able to translate any incoming or outgoing character to assure compatibility between SuperPET and any telecom partner.

In summary, I can recommend the PETCOM package to those who'd like to explore SuperPET's communication capabilities fully. You'll learn a great deal about our machine in the process from a very good program and fine documentation. [Ed: the last price we've seen on PETCOM is $98.95 Canadian. Write Ph.D. Associates at the address given in this article, or call 416 667 3808. The disk is available in both 4040 and 8050 format. VISA, check, or money order accepted.]

BITS BYTES & BUGS by Gary Ratliff, Sr.

215 Pemberton Drive, Pearl, Mississippi 39208

What is the problem with the following assembly language segment?

```
ldd text
cmpd #0
if eq
  ldd #error
  jsr printf_
endif
```

If you didn't notice that it contains an unneccessary test then you are ready for our tutorial on the uses and abuses of the structured language constructs allowed by the Waterloo Assemblers for both 6502 and 6809 code. The cmpd #0 is redundant because the condition is tested by the 'if eq' construction automatically.

In this installment we'll examine closely the structured programming statements allowed by the 6809 assembler in an effort to understand just how these structures assemble code. I hope this'll keep you from creating redundant code because you recognize the branches and tests which the Waterloo Assembler generates automatically.

In all examples, the LABELS to the left of the code are added by me so you can follow how the assembler converts the easy-to-write structured code of SPET'S assembly language to 6809 mnemonic code. To appreciate what the assembler does, refer to the much-cited Software Engineering for Micros by Ted Lewis. There he shows the code you would have to write to create a structured program in assembly language--if Waterloo had not provided the structure for us.

In the examples, I use capital letters and lower case interchangeably. The CAPS are for emphasis. My text follows the order in which the manuals discuss concepts, found on pp. 145-152, 6809 Assembler Manual, and pp. 129-135, 6502 Assembler Manual. In addition, I show the names of subroutines to make it easy to follow the material. The real assembler code would use hex addresses.

Throughout this discussion, -c will represent a condition not c. Thus if the test condition is 'eq' then -c means 'not equal', and if the condition is 'cs', then -c means carry clear.

First, let us explore the construct: if/endif. How this appears in source code is shown in the left column, and how it is assembled is noted in the right col-

umn. I'll use the same method throughout this article. Remember that 'putnl_' does a carriage return and a linefeed to the screen, so when we encounter a CR in the string we manipulate, we call the system routine which prints one.

```
lda char                LDA CHAR    ;Note that the source code 'if eq' is
cmpa #13                CMPA #13    ;revised by the assembler to BNE, or
if eq                   BNE  skip   ;BRANCH IF NOT EQUAL, to the label SKIP
  jsr putnl_            JSR PUTNL_  ;under condition -c. I show the location of
endif                               ;the psuedo-label SKIP for clarity.
lda #12           skip  LDA #12
```

Here we see that the 'if eq' structured language construct generates code which branches on the opposite condition to an artifical or psuedo-label called SKIP. SKIP is attached to the instruction which follows the 'endif' construction. The 'endif' by itself generates no code. It merely serves as a point of reference for the artifical label 'skip'. This label attaches to the instruction following 'endif.' Next, let's look at the if/else/endif construct:

```
lda char                LDA  CHAR
cmpa #13                CMPA #13
if eq                   BNE  skip          ;if condition -c, go to label SKIP
  jsr putnl_            JSR  putnl_        ;otherwise, execute the code in-line,
else                    BRA  exit          ;in the sequence shown.
  jsr putchar_    skip  JSR  PUTCHAR_
endif
lda bhar          exit  LDA BHAR
```

Here we see that the assembler code uses two locations, labelled SKIP and EXIT. Skip is defined as the location immediately following the 'else' clause, which generates a BRA in 6809, or in the 6502 a JMP instruction to the instruction immediately following the 'endif.' The condition (BNE) is again the opposite of that of our source code test. At this point, I suspect beginners understand why it is so much easier to write code in Waterloo's structured assembly language than to write direct assembly-language code.

Let me point out that if in writing our program we define labels which will be generated automatically by the assembler, we create redundant code. The example below shows what can happen. We've written in one label, EX1, and the assembler in effect creates a second, EXIT. Our EX1 label is totally redundant.

```
      if eq                       BNE EXIT       ;1st BRANCH instruction
        jsr putnl_                JSR putnl_
        bra ex1                   BRA ex1        ;2nd BRANCH instruction
      endif
ex1   lda  #12            EXIT  EX1 LDA  #12
```

Now you get the idea. Structured programming statements take the risk of error out of programming. Those who are used to writing assembly language code may be more prone to duplicate the efforts of the assembler; old hands beware!

The next construct is 'loop-endloop', which is easier to follow than the guess/admit/endguess structure. Consider the example following:

```
loop
  lda  ,x+          START LDA ,X+
  quif eq                 BEQ  exit     ;BRANCH IF EQUAL--if c is met--to exit.
  jsr putchar_            JSR putchar_  ;The 'quif' condition is met when the
endloop                   BRA start     ;value loaded in A register is 00.
lda test          EXIT    LDA TEST
```

The loop construct creates an artificial label START, defined as the next location after the loop instruction. An EXIT label is also defined, and found as the instruction following the endloop construct. The endloop construct likewise generates a 'BRA Start' instruction. Thus a loop/endloop structure without any internal 'quif' or a valid 'quif' can create an infinite loop--which will not only anger system operators but also create a monstrous connect-time bill when you are online with a mainframe. [Note also that a 'quif' instruction generates code to branch on condition rather than a branch on 'not' condition, which an 'if eq' in structured code would generate.] Next, we try a loop/until construct:

```
loop
  lda char          START  LDA char
until eq                 BNE start  ;BRANCH IF NOT EQUAL on -c; loop to start.
ldb test          EXIT   LDB test   ;We load A register until it gets 00
```

Here, the 'until' condition translates into a BRANCH ON NOT condition to label START. The artificial label EXIT is reached only when the condition tested for is found to be true (when we load 00 in A register).

Last, let us look at the forms 'guess/endguess' and 'guess/admit/endguess.'

```
guess                              ;The guess/endguess structure by
   lda test       LDA TEST         ;itself generates only inline code.
endguess

guess
  cmpb #test      CMPB #test
  quif ne         BNE  EXIT        ;if -c, leave the structure for the FIRST
  lda  #nota      LDA  #nota       ;code line following endguess. Note that end-
endguess                           ;guess writes no code.
lds    ,x    EXIT LDS  ,X
```

An artificial label of EXIT is attached to the instruction following the 'endguess' construct. The 'quif ne' generates a BRANCH ON NOT to EXIT. Again, note that 'quif' generates the the same conditional branch in assembly code as in the structured code, while 'if cond' generates code to BRANCH ON NOT condition.

```
guess                                  In this example, we use a CASE structure
  ldb  test          LDB test          to determine what value will end up in A
  cmpb #1            CMPB #1           register. We compare the value in varia-
  quif ne            BNE a_label       ble TEST with 1, 2, 3. If we don't find a
  lda #5             LDA #5            1, 2, or 3, we default to a value of -1
admit                BRA EXIT          in the A register. But--if we find a 1 in
  cmpb #2    a_label CMPB #2           TEST, we put a value of 5 in A register;
  quif ne            BNE b_label       if we find 2 in TEST, we load A with 6;
  lda #6             LDA #6            if we find a 3 in TEST, we load A with 7.
admit                BRA EXIT          Note that if any ADMIT clause is found to
```

```
  cmpb #3     b_label  CMPB #3
  quif ne              BNE c_label
  lda #7               LDA #7
admit                  BRA EXIT
  lda #-1     c_label  LDA #-1
endguess
  sta vary    EXIT     STA vary
```

be true, we EXIT guess/endguess at the next line of code. Guess/endguess generates a BRA, or in 6502 a JMP, to the line following the the endguess. This line is assigned an artificial label EXIT. Each of the 'quif' instructions generates a BRANCH ON NOT CONDITION to the next ADMIT statement, identified in the example as a_label, b_label, and c_label.

Guess/admit/endguess is a multi-choice test. We could have presented a menu with several valid choices. The guess/admit/endguess lets us see if the value in TEST is valid, and implement it, or set a -1, so we can correct whatever is wrong.

You should now understand exactly what code will be generated by the assembler when you use structured programming statements--and this knowledge should enable you not to duplicate the effect of these statements in your own code.

COPYING...(YAWN...BZZZZ) Having a lot of copying to do from disk to disk, we got curious (yawn) about why it took so long and ran a test or two. The results startled us. We copied the same APL program file from disk to disk five times and averaged the results. Even the mBASIC program written to copy WCS PRG files was faster than the mED 'copy' command. The time difference to copy SEQ files was similar. If you sleep whilst at the keyboard, you know how to get it. We prefer to copy with g ieee8-15 and to sleep in bed.

Copy Method:	Time (sec):
mED copy command	43
mBASIC program copy	33
microPIP copy cmd.	19
g ieee8-15. copy cmd.	14

We stumbled into this while learning to convert Waterloo PRG files into printable files we could read in mED. The program below copies PRG files exactly, and as <u>string</u> values, with the LINPUT statement. Yes, the copied files run. Obviously you can parse a$, convert it to ordinals, and read the PRG file in either hex or decimal--in the mED. Which is what we were after. It beats a GET hollow.

```
100 ! copyas$:bd
110 on eof ignore
120 endd$=rpt$(chr$(0),4)+chr$(2)+chr$(0)  ! Ends all assembly language modules.
130 open #20, "filename,prg", input : 1%=1
140 open #30, "disk/1.filename,prg", output
150 loop
160   linput #20, a$
170   if idx(a$,endd$) then flag%=1%
172   if io_status<>0 then flag%=1%        ! Ends all other 6809/6502 PRG files.
180   if flag%
190     print #30, a$;
200   else
210     print #30, a$
220   endif
230   if flag% then quit
240 endloop
250 reset : stop
```

[Note: we don't know what Waterloo did with the COPY command in mED, but the code must get to the barn by going around a couple of mountains first. We suspect there's a lot of checking and acknowledgement going on.]

KING KONG, GODZILLA, AND NOW: THE MONSTER FROM THE MONITOR

Ye ed crept into the monitor and spent about a month in that often lightless cavern, in dark pursuit of an assembly language alphanumeric sort for the 6809--and emerged with 1) a sort, 2) a long, hairy coat of green monitor mold, and 3) 10 pounds lighter (long overdue). We've pleaded for six months for some talented bit-twiddler to write such a sort, but nobody volunteered (for reasons which soon became obvious). On the Utility disk which we define this issue, you'll find 3 versions of ALPHA (better we should call it NEMESIS): one which uses all of SuperPET's memory, and will sort 1585 strings into alphanumeric order in 20-25 seconds, and a short version, which you can SYS from language, which will sort 300 strings in 2 seconds. The third is a 300-string version which runs in the monitor. So far as we know, this is the first 6809 assembly-language sort to become available (and the last we'll write for a while). It puts capital and lower case letters in the same alphabetical order (MICRO and micro are equivalent); gets the list to sort from disk and files the sorted list to another disk file.

* * *

SPMON and EXMON

Terry Peterson employs the phrase 'bit-twiddler' to distinguish those of us who indeed twiddle bits in assembler. SuperPET's structured assembly language is a delight to use, but debugging and analysis of pieces of your programs is slow going in the standard Waterloo monitor. On the ISPUG utility disk we have two extended monitors: SPMON and EXMON, the first written by the grand sachem of our tribe, Terry Peterson, who wrote HESMON for the VIC; and EXMON, written by our own Redoubtable Gary Ratliff. It is fortunate that they extend the monitor in two different ways. SPMON is a powerful program, which loads from main menu. You can execute all DOS commands from it; load and save program modules, slowstep or quickstep through a program, compare any program with any other (or part of memory); set breakpoints; calculate in hex, decimal, or binary, load a program off disk anywhere into memory (including the banks); 'hunt' for a phrase or number, and much more. We would have spent many months writing ALPHA were it not for SPMON.

EXMON takes a different approach. While it will load and save modules, and executes all Waterloo monitor commands as-is, it also provides both 1) a number of extensions similar to Terry's, and 2) the 'hooks' for you, the programmer, to add features you want. Gary included Avygdor Moishe's 'linker', so you have a built-in way to add code (Avy is a talented bit-twiddler from TPUG who wrote PET-COM, a TC program for SuperPET, and some of the other goodies on the utility disk). Dedicated bit-twiddlers will love this one. Gary added a RESTRICTED level to EXMON, which you cannot enter until you crack the code for entry--using EXMON, of course.

* * *

FOR NON-BIT-TWIDDLERS

Lest you think the entire utility disk is devoted to bit-twiddler specials, be advised it is not. The remainder of the programs are utilities useful for novice and expert alike.

Some have asked why so many ISPUG disks are offered in recent issues. Simple: we have more stuff available than ever we can cram into the Gazette, and there is no other way to make it available to you. When you read the partial directory below, you'll see what we mean. We've kept our promise to document disks, too!

All source files are on disk, though many are not shown below to save space. In addition, all disks include programs not shown, again to save space here. The partial directory below will give you an idea of what our members have put to-

gether. Everything on disk works. Since all source files are available, you can modify any to your own wishes. For those who don't know assembly language: most programs adapt to any printer as is. Where they won't because of printer differences, you can change ONE line; the change is explained. After you've made it, if you can still 1) breathe and 2) type, follow the procedure on page 143, Vol.I of the Gazette to reassemble and relink. It's simple. We add that the 4040 version (2 disks) we had to cram in, and the 8050 has about 100 blocks free.

```
116  "contents:e"          SEQ   Instructions on programs; table of contents.
21   "spmon:men"           PRG   Extended monitor; loads at $6000. T. Peterson.
21   "spmonlo:men"         PRG   Same extended monitor; loads at $2000.
42   "spmon.doc0:e"        SEQ            Instructions thereon.
18   "hello:men"           PRG   A modification to SuperPET's operating system,
plus supporting programs         which lets an interrupt-driven routine reside
and all source files.            in the switched banks. HELLO loads GSCROLL,
  hello and gscroll              in bank 15--a dump to disk or printer, and a
  by Terry Peterson.             routine to stop screen scrolling at any time.
                                 GSCROLL also provides 'instant phrases' on the
7    "gscroll"             PRG   shifted keypad (you can turn them off or on).
17   "A000_AFFF"           PRG       ROM images of all ROMS 6809-side.
17   "B000_BFFF"           PRG   These have been cross-checked on 4 machines. If
17   "C000_CFFF"           PRG   you suspect problems in ROM, compare your ROMs
17   "D000_DFFF"           PRG   with these images, using SPMON. Instructions on
9    "E000_E7FF"           PRG   how are included in: spmon.doc0:e
17   "F000_FFFF"           PRG

66   "xmon6809"            PRG   Gary Ratliff's extended monitor,
80   "instruct_exmon:e"    SEQ   and the instructions.
3    "graph_index:e"       SEQ
19   "bar_graph:e"         SEQ   Delton P. Richardson's bar-graph program, tutorial
26   "bg_tutorial:e"       SEQ   and programs. See graph_index:e for details.
74   "bgmenu:bp"           PRG   Some supporting programs and examples not shown.
8    "all_cmd_files:e"     SEQ   All .cmd files for all programs on this disk.
3    "adump.mod"           PRG   Print any SEQ file to printer from main menu--
      recent revision allows use of any printer, and gives an optional linefeed.

3    "alpha:6000"          PRG   ML sort for 300 or fewer strings. Monitor version.
17   "alphabig:0a00"       PRG   Same sort, but for up to 1585 strings. Monitor.
7    "alphasys:6000/6"     PRG   Same sort. SYS from any SuperPET language.

2    "chgadrs.mod"         PRG   Change device number on disk drives from menu.
11   "ddisk.asm"           SEQ   A do-it-yourself package; conditional assembly.
1    "diablo.mod"          PRG   Margin set, main menu, DIABLO or COMMODORE 8300P.
19   "loader:au"           PRG   Jim Swift's alphabetizing APL loader.
40   "mdir.asm"            SEQ   Send 2 column directories to screen/printer. For
4    "mdir:f9000"          PRG   all printers; optional linefeed. Monitor version.
24   "nscroll.asm"         SEQ   A dump to disk/printer. Also stops scrolling at
2    "nscroll.mod"         PRG   at any time. Great in PIP and in APL.
39   "pdir.asm"            SEQ   Sends 2 column directories to screen/printer from
4    "pdir:men"            PRG   main menu. All printers; optional extra linefeed.
1    "reset:men"           PRG   Resets from main menu to whatever is in upper 64.
1    "retrieve:men"        PRG   Recovers lost programs and languages.
21   "save.my.text.asm"    SEQ   Saves text in memory after an accidental exit from
2    "save.my.text:men"    PRG   language, to disk or to printer.
```

2	"udumlf:men"	PRG	Menu loaded UDUMP; all printers; extra linefeed.
2	"udump:men"	PRG	" " " ; all printers, without linefeed.
2	"ulfmon:7e60"	PRG	UDUMP loaded from monitor, with extra linefeed.
2	"umon:7ef0"	PRG	" " " " , no extra linefeed.
19	"when.asm"	SEQ	Time and date set from main menu; you may bypass
2	"when:men"	PRG	one and set the other or correct errors.
8	"COPIER.8050.BP"	PRG	Gets data from bad disks. (4040 version on 4040)
34	"med_v1.3"	PRG	A Toebes special. Version 1.3 of mED. FAST!!!!
3	"ml.loader:bp"	PRG	Program to load an ML module from mBASIC.
7	"TRANSLATE.BP"	PRG	Translates 6809 SEQ files for WordPro.

Those who want a copy of the ISPUG Utility Disk: send $10 U.S. for the 8050 version, or $16.00 U.S. total for two 4040 disks. Get 8050 from the Editor, PO Box 411, Hatteras, N.C. 27943--or 4040 from the Secretary, 4782 Boston Post Road, Pelham, N.Y. 10803. Make checks to ISPUG. Postpaid. Overseas, add $2 U.S. for additional postage.

◇◇◇

PRINTERS AND DRIVES From data on the membership application forms we can report the distribution of printers and disk drives. We do wish everybody had reported; we need the information to guide us on what gets printed. On drives, the 8050 is the clear favorite: 70.2% of members own it; 4040 is second with 24.5%; the rest are 2031 and 8250. Some 31% of members use the 8023 printer; 20% the 4022; 17.5% the MX80. The letter-quality printers (DIABLO, 8300P, Spinwriter, etc.) are owned by 13.1%; the rest of the printers are not found in large numbers.

◇◇◇

A SPEEDY MED John Toebes and his talented fiance, Mary Ellen Kelley (also an assembly-language programmer) have rewritten the V1.1 mED. Their version 1.3 is on the ISPUG Utility disk. We haven't used Waterloo's V1.1 since V1.3 arrived, and we can find no bugs in it. The new mED is so fast it could qualify for the first row at Indianapolis...and John and Mary are now at work on V1.4, which removes the old V1.1 bugs and adds some bells and whistles.

◇◇◇

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

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

Send check to the Editor, PO Box 411, Hatteras, N.C. 27943. Add 30% to prices above to cover 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)

Name:______________________________ Disk Drive: _________ Printer:_____________

Address:____

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

[] Check if you're renewing; clip and mail this form with address label, please.

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: Paul V. Skipski, Secretary, 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. Secretary, ISPUG: Paul V. Skipski. Editor, SuperPET Gazette, Dick Barnes. Send membership applications/dues to the attention of Mr. Skipski; newsletter material to the attention of Dick Barnes, Editor. SuperPET is a trademark of Commodore Business Machines, Inc.; WordPro a trademark of Professional Software, Inc. If you send inquiries, enclose a self-addressed, postpaid envelope (4 x 9.5 inches, please). If you submit material for the Gazette, enclose a suitable return/reply envelope, postpaid. Canadians: enclose Canadian dimes for postage. See enclosed application form for membership dues. The Gazette comes with membership.

For all outside the U.S.: All nations members of the Postal Union offer certificates good in the postage of any other country for a small charge. The Union includes most nations of the world.

FIRST CLASS MAIL

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

First-Class Mail
in U.S. and Canada