SUPERPET GAZETTE

First, the bad news. Associate Editor Steve Zeller, who has conducted our APL column since it started, suffered another hardware failure in his SuperPET and decided to move on to a new IBM PC AT (he has it, but the bank owns it). We have been expecting this, for Steve's thoughtful employer provides him an older IBM PC at work. With many regrets and much praise for his labors in behalf of all APLers in ISPUG, we wish Steve good luck, and suspect he'll have it--his old printer, cabled to the PC, went to work at once with no hardware changes, a good augury indeed. We'll miss a fine writer, a sharp mind and a good friend.

And the good news: Reg Beck, of Williams Lake, B.C., a charter member of ISPUG, has agreed to step in as the new APL Associate Editor. His first column appears in this issue. Because Reg teaches APL each year to a new crop of students, he understands the problems the language poses for newcomers--of whom we have a bunch, particularly in schools. We therefore asked Reg to review the aspects of APL which often puzzle those new to the language as implemented in SuperPET.

And some bad news: The lead article in issue 14, Volume I, announced that Waterloo Pascal and Structured Basic are available for the C-64, which Commodore now sells to schools in a SuperPET case with a green screen. We speculated that this might mean the end of SuperPET and 8032 production. It appears we were right; Commodore has told its dealers that when the limited present supply of SuperPETs is gone, Commodore will produce no more. The 8096 has been discontinued; upgrade boards to convert an 8032 either to SuperPET or to an 8096 likewise are no longer being made. We suspect Commodore may change its mind if the schools raise a fuss--but only if.

We've expected the news for six months or more, as Commodore moved to support the educational market with the C-64. We see no signs that FORTRAN, APL, or COBOL will be available on the C-64. The schools which teach those languages on SuperPET must, therefore, either buy more SuperPETs now, while they are still available, switch to computers far more expensive than SuperPET, or forever hold their peace. If, as we suspect, a number of schools are committed to SuperPET and to courseware for it, we'd suggest they send the Chairman of the Board of Commodore a letter from their principal, superintendant of schools, or from the school board. We frankly doubt the C-64 can hack all languages the schools want to teach. Waterloo had to shoehorn subsets of most languages into SPET's 96K; what can be done with the C64's 64K? We suspect Commodore knows this; if so, it must either keep going with SuperPET or come up with a better answer--or Apple and IBM (PC Jr.) will gobble up the computer science market.

BLOODY NOVEMBER

We'd guess everybody subscribes to the Gazette in the fall, after they comb out the beach sand, come down from the mountains, and return to business or school. Almost a third of you will find your subscriptions expire with this issue. If your address label is underlined in red, your membership in ISPUG has expired.

If so, check the RENEW block on the last page and send it with your address label or a copy. Remit $15 U.S. in North America or $25 if elsewhere. Please do it before we scratch your label from the disk file and must retype all that stuff!

We doubt that Commodore built a drop-dead switch into SuperPET so that the poor thing will expire when production ceases; all chips used are available from commercial parts stores, except for imaged ROMs. Because we have disk images of those on the ISPUG utility disk, we'd reckon present SuperPETs can be maintained for some time. Our SuperPET does just about everything we want done; we're going to keep it until a machine far superior to anything we see on the market becomes available. We aren't going to settle for rodents, icons, and darling tiny disks at $4.50 each, nor tolerate the people-hating operating system of the IBM PC and its clones.

Meanwhile, many schools need language manuals. We recently discovered that many of them are no longer available from Howard W. Sams, the printer, and so called Waterloo. Most of the manuals are available from WATCOM Publications, 415 Phillip St., Waterloo, Ontario, Canada N2L 3X2. Waterloo said that the quantities at left were available. We also talked to Tom Dow, Commodore's SuperPET manager, a knowledgeable and helpful guy. He's arranged to make available--directly from Commodore--either of two different packages: 1) Manuals for all of the languages, plus update pages for V1.1, plus the COBOL manual, plus V1.1 disk software, and 2) a V1.1 update package which holds update pages for all manuals, a COBOL manual, and V1.1 disk software. Because Commodore has a matched number of manuals and disks, it will not sell manuals or disks individually. You may order the packages only. If you want single manuals, order from Waterloo. This leaves only one problem: Waterloo is out of COBOL manuals for SuperPET. Waterloo did say, however, that the V2.0 COBOL manuals (for the IBM PC) are available and suffice to learn the language; we're told there are few differences between the V1.1 and V2.0 implementations, except for three pages on system dependencies in the SuperPET manual (which are covered in the COBOL tutorials on disk anyway). Order V2.0 COBOL manuals from Waterloo for $15 (U.S. in the U.S., Canadian in Canada). What do you do for mBASIC manuals? Weep or (horrors) X---x it?

Quantity	Manual	Price
2000	APL	9.00
200	Assembler	10.00
500	mFORTRAN	10.00
780	mPASCAL	10.00
0	mBASIC	.....

As we go to press, we do not have prices or part numbers on the Commodore packages. If the information arrives in time, we'll add it to the last page of this issue. Check there.

THE COMPILER BRIGADE PROGRESSES Well, we may yet have a compiler for microBASIC. John Toebes' six slaves have finished disassembling all of mBASIC and most banks have been successfully reassembled. The hard work of commenting the code (ugh!) is well underway. We list the slaves at left, and suspect all of them are, by now, well aware of the basis of the old Army rule: "Never volunteer!". After having disassembled a few bytes of Waterloo code ourselves, we suspect all of 'em are due jump pay (if you find ten consecutive lines of code which don't jump or branch to another piece of far-distant code, you stop before you ruin a good day). Our thanks to the six for much hard work; once the comments are finished, our genius John Toebes will then write the compiler. Uncle Sam snapped him up, however, for a four-year tour in September, and with superb wisdom sent him to Washington, D.C. (on a 2nd Looie's pay, he will there starve to death after he pays the rent). With more wisdom, the AF then assigned this starving fella, trained as a system programmer, to maintaining old programs (COBOL, John?). Worse yet, John had to leave new bride Mary Ellen in Raleigh to finish school. Oh, well; if you don't like your work, love and hunger will take your mind off that problem.

Loch H. Rose, Cambridge, Mass.
Louis Mittelman, Gordonsville, Va.
James M. Sweeny, New Paltz, N.Y.
Alain Proulx of L'Ephiphanie, Quebec
Brad Bjorndahl of Bramalea, Ontario
Russ McMillan of Madison, Wis.

ONCE OVER LIGHTLY Miscellany A note from Austin Hook, president of the Computer Shop of Calgary: "Congratulations to the Calgary School Board for their decision to purchase another 81 SuperPETs to be installed by September 1st. They are also upgrading some of the 8032s which they own to SuperPETs, bringing the total in the system over the 100 mark. We agree the SuperPET is still by far the best machine for Computer Science instruction. No product on the market can touch its price/performance." Yes, indeed. But does Commodore agree? If schools want SuperPET to stay in production, scream!

ON BEING A TERMINAL AND HOSTCM Longtime member P.J. Rovero, about to depart for Italy with wife and newborn, reports: "I'm very busy with work on my thesis; the SuperPET gets a workout, but these days it's mostly just a terminal. I use NEWTERM going to/from the Control Data and Cyber mainframes at one site and just the mED with the IBM running HOSTCM. I'm very pleased with version 1.3 of the mED. No bugs detected in any area, including host communications. [Ed. V1.3 of the mED, in optimum hand-writ assembly code by John Toebes and wife Mary Ellen, is on the ISPUG Utility Disk.]

"A note for other HOSTCM users--our facility recently upgraded its mainframe operating system. The system prompt and response characteristics were not changed ($11 and $13, as per p. 60 of Systems Overview manual), but now the first character after response is always a "delete" ($7F). Apparently, when the first character of a system message is "delete", the checksums are not properly translated. The host and SPET then trade negative acknowledgements all day....

"The fix is to specify $7F as the response character when SETUP is invoked. End of problem. SuperPET users were not the only ones affected by this change. At the Naval Postgraduate school, HOSTCM is the system-supported micro-host link. So even CP/M and MS DOS users are using HOSTCM through software developed by members of the school's Hobby Computer Club." [Ed. Which means, of course, that the software built into SuperPET for Host communications must be written for the other computers. If IBM finds out about this, heads will hang in Armonk.]

COST OF BACK COPIES Gee, we got this stern letter from a member, protesting what he calls "The outrageous price for back issues of the sheet." Let's take a look at the price of a 30-page issue, which is $3.75. We can't stock back issues in quantity unless we buy a warehouse, so we have to print them as they're ordered. How? By hiring somebody to come in and run off the copies, ten sets at a time, on a Xerox machine--and the printing runs us 10 cents a page. Cost of printing a 30-page issue is $3.00 plus the cost of labor to print, collate, and staple, plus an envelope for multiple copies plus first-class postage (we figure the folks want the copies now, not later). Add it up. Getting rich we aren't.

8050 DISKS ON AN 8250 DRIVE We continue to get notes saying "Geez, that 8050 disk you sent me gives an error message on my 8250...." Sure it does. Different BAM. First time you try to get a directory or load a file, you get an ERROR 66, ILLEGAL TRACK OR SECTOR message. Ignore it and try again. Lo, no more problems. Best COPY the 8050 to a formatted 8250 disk. BACKUP won't work.

RENAMES ON DISK/1 It's easy to overlook the fine print in Systems Overview, and to try to rename a file on disk/1 with "rename disk/1.oldname to disk/1.newname", only to find (sob!) that "disk/1" is now part of the new filename. Don't. Use "rename disk/1.oldname to newname," or else thee must cudgel "disk/1" out of the new filename with a monster like "rename disk/1.disk/1.newname to..." Okay, Fred Foldvary, the unwary are now warned of the trap you fell into.

HALGOL!!!! Out in Orange County, California (where anybody with a Democrat for a great-grandaddy is considered a rascal and a radical) we found new member Hal Hardenbergh, who builds special computer boards, most equipped with a Motorola MC 68000. Hal does business under the name of Digital Acoustics; he's a speed merchant; among other endeavors he's writing a BASIC-like new language called HALGOL. We'd say it's about half-done. Associate Editor Terry Peterson has a version hooked up to a 68000 board attached to his SuperPET. HALGOL, compiled as each line is written, is designed for superfast and accurate math, even with transcendentals (infinitely repeating numbers, such as pi). The little program at left must calculate a bunch of floating-point values for tan(i) which, if done with absolute accuracy, would show no error when Root Mean Square error is determined. Absolute accuracy is, of course, impossible--especially when computed in binary. The RMS error is a measure of how accurate the transcendental routines are in any computer. How good is HALGOL? Well, the program at left, iterated 2500 times, runs in 18 seconds with an RMS error of 4.1006 E-12. On our SuperPET, in mBASIC, it runs in 780 seconds for an RMS error of 6.1153 E-7. Note that the difference in RMS error is roughly five orders of magnitude, and that the program executes 43 times as fast in HALGOL as in mBASIC. Some language, and some microprocessor! Number crunchers and speed freaks, watch HALGOL.

```
for i=1 to n
  b=tan(atn(exp(log(sqr(i*i)))))/i-1
  z=z+b*b
next n
z=sqr(z/n)
print "R.M.S. error=";z
```

Hal bought his SuperPET so he could wire in a black box and power supply to run HALGOL off the 6502 side and hitch up the 68000 processor, which runs at 12.5 MHz. Yes, he plans to offer the black box to those who want it, together with HALGOL, when done. More later, when we have prices and performance in hand. In the meantime, if you care to sample demon Hal's wit and wisdom and follow the progress of HALGOL, send $15 for a year's subscription to his newsletter, DTACK GROUNDED, which arrives about once a month. But don't write Hal any letters. He won't answer 'em, being far too busy making a living. The newsletter is witty, full of gossip and incisive insights on the computer industry. Send the loot to DTACK GROUNDED, 1415 E. McFadden, Suite F, Santa Ana CA 92705.

STAPLE BONUS Our new print shop drove enough staples into the last issue to hold back the Mississippi at full flood. We know who did it; when he goes for his last swim in 1984, we're gonna staple his shorts to the bath-house door with a full clip. Those who opened the last issue with pliers will be avenged.

WE LOSE FAITH FROM TIME TO TIME That the voters in a democracy are informed, that is. Though we announced last issue that Paul Skipski, former Secretary, had resigned, we still get letters addressed "Dear Paul." Oh, well; just last week a letter arrived at the White House to "Dear President Garfield." Let us give whoever wrote it credit for knowing that General Grant is out of office.

SAVE TO TERMINAL Since SuperPET handles the keyboard and terminal as files, you can accomplish some strange feats. Loch Rose writes that you can "list" a whole program to screen, with no pauses, with a "save terminal" in mBASIC; anybody want to guess what happens if you "put terminal" in other languages?

OOPS and AHEM DEPT. Any Okidata owners who tried to run the function dump we printed last issue (I, 274) found it won't work until line 11 reads as shown at left. We goofed when we entered it by hand from hard copy and couldn't test because we don't have an Oki.

```
[ 11] LOOP: SCR←IO
```

WEST GERMAN INFO BOX A. Pietrzok of Frankfurt writes that he's searching for other SuperPET users in Europe and America and says he leads a SuperPET "Information Box", tel. 06181 / 48884. If interested, call or write to Friederichstr. 5, 6000 Frankfurt/Main, West Germany. He now knows about ISPUG.

8032 IS NOT NECESSARILY AN 8032 Letters keep coming in saying "I have this telecom program which works fine on an 8032, but it won't work on SuperPET in 8032 mode..." Well, the 8032 in SuperPET is equipped with a 6551 ACIA, and plain 8032s aren't. That's why the Commodore public domain bulletin-board software had to be modified for SuperPET--and why programs for the serial port on plain 8032s won't work properly when used on the 8032 side of SuperPET. What to do? Send $20 for "The SuperPET Serial Port," to WATCOM Products, 415 Phillip Street, Waterloo Ontario, Canada N2L 3X2. Learn how to handle the ACIA, which controls the port. Or use the programs on the ISPUG master telecom disk, which cope with the ACIA.

PRINTER FILES FOR THE mED In Vol. I, issue 13, p. 220, we explained, but not clearly, how to send disk command files to printer from the microEDITOR. We've had a lot of questions since, so we restate the fundamentals. If you want to be able to send your printer a command to change its format (margin, character set, etc.), write the command sequence in program in your favorite language, and file it to disk. Then <u>copy</u> the disk file to printer from mED. Suppose, for example, you want to set your printer for boldface, and that the command sequence is ESCAPE 12. In language, use a program like that at left, which writes the proper ESCAPE sequence to disk. When you want boldface, "copy boldface to ieee4" in the mED from disk. Be <u>utterly certain</u> that any CR is suppressed with a semicolon, as at the left, unless a CR is <u>demanded</u> by the ESCAPE sequence.

```
open #12, "bolface", output
print #12, chr$(27)+"12";
close #12
```

Those with Commodore printers often must use secondary addresses to change printer parms. You can't OPEN and CLOSE secondaries from <u>disk</u> files in the mED. But--you can send the commands to secondary addresses directly. Suppose your printer manual directs that you open secondary channel 13 and then to print to it, as at left. The program sends to secondary address 13 only one CR. You can do this easily from mED. Open up a blank line in mED, leave the screen cursor on it, and at command cursor say: ". p printer4-13". The "." says "transmit only the line the screen cursor is on", and the rest of the command prints a null line--plus a CR! The CR is sent to the proper secondary address at printer. If the printer command requires any character you can send from mED (any ASCII code above 32), you can send that, too--by putting it on the single line you "put" to printer. Remember that a CR is always sent by mED at end-of-line by any "put". Last, you have the option to "copy" any disk file with the right printer commands in it to "printer4-x" (where "x" is the proper secondary address). Try <u>that</u>, Arnold. Your printer should swoon.

```
open #12, "printer4-13", output
print #12
```

ADA 1450 INTERFACE We printed this tip on switch settings on the ADA 1450 IEEE-Serial Interface long ago, when the world was young. Disregard the recommended switch settings Commodore publishes and which the maker recommends. They will not work from SuperPET to any serial printer addressed as "ieee4". The only switch settings which will work universally, in WordPro, PaperClip, 6502 and 6809 are: switch 1 ON, switches 2 through 4 OFF. All BASIC 4.0 listings are sent in caps; everything else prints as "what you see is what you get."

HOW TO DRIVE MONITORS FROM SUPERPET Don Momberg of Green Brook, N.J. writes that he'd love to be able to drive a bunch of student monitors (not other SuperPETs, but plain monitor screens)--or a wide-screen projection TV from SuperPET. He says he's seen it done from Apples and Ataris. If you'd like this capability, write the Editor. If enough people are interested, we'll look into it.

AN INDEX TO THE ISSUE We hope to start indexing each issue of Volume II, of which this is issue 1. Look on the very last page (the outside cover). If we succeed, you'll see it there; if not, we'll regroup and try next issue. It's a bit hard to index an issue by page number when printed it isn't yet, but we must try--even ye ed can't find things any more yet.

THE NAME OF PAPERCLIP Bob Davis writes that his dealer kept sending him the wrong version of PaperClip, and that there are three: PaperClip 64 (for C64), PaperClip (for the 8032), and PaperClip Expanded (for SuperPET and 8096). When Bob finally got the nomenclature straightened out, he got the right 'Clip. If you order, be warned. The mad people who prepare parts catalogs and name parts live in their own world: we once tried to order a plain 8-inch crescent wrench, and after a month found it named "Tool: wrench, adjustable, offset head, worm driven." Care to guess the proper nomenclature for a roll of toilet paper?

UD11 TOWER IN BACKWARD Those with old three-board SPET's may have had someone install the two extra switches which control sockets UD11 and UD12. When Bob Davis installed his 'Clip ROM in UD11, SPET died. Turns out that the tower or socket for UD11, installed with the switch, was in backward. When Bob reversed the ROM, all was okay. Watch it. (On two-board SPETs, install the 'Clip ROM in U46, upper board.)

A BUG IN NEGATIVE INTEGERS
by Associate Editor
Stan Brockman

Two ISPUG members, Paul Schain and Peter Frisk, recently pointed out another bug in mFORTRAN's formatted input. It seems that negative integers cannot be read back from a file with the field specification used to write the file in the first place! The little program at left writes a record containing five one-digit negative numbers to a scratch file, using I5 format. The record looks like this, with ^ showing a blank character in the file:

```
   integer k(5)
   open(11, file="scratch")
   write(11,1) (-i,i=1,5)
1  format(5i5)
   rewind(1)
   read(11,1) (k(i),i=1,5)
   print,k
   close(11)
   end
```

```
^^^-1^^^-2^^^-3^^^-4^^^-5
```

The problem is that mFOR formatted input expects the sign, if present, to the in the first position in the field being read (-^^^1, etc.); if it's not there, a conversion error occurs--which is exactly what happens when the record we just made is read with an I5 format. You can, however, read the file with a "3X,I5" format. Well, of course you don't know how many digits exist in any integer you read, let alone whether a value is signed or not! The language should let you write formatted input instructions which are flexible enough to specify the maximum field width to be read without concern for where the sign is. After all, you can specify formatted input with REAL data (using the "F" and

"E" descriptors) and not be fussy about the position of the sign; the data are read correctly.

You can sidestep the bug in three different ways: 1) Read the data with an "A" descriptor of the required field width; then convert it to integer with the intrinsic function "cnvc2i". For the example above, you read with an "A5" format; 2) Read the data into a real variable with the "F" or "E" descriptor, specifying zero decimal places; then convert it by setting it equal to an integer variable (i.e., read with "F5.0" format); or, 3) Use list-directed (LD) reads. Of course, you must change output to disk to include comma delimiters between data values; if you do this, an LD read will properly and swiftly read all values, including negative integers.

Alternative 3 appears to me to be the most reasonable solution if you can save current data to be read later. The other two will save your bacon if you want to read data you saved last month or last year. (Newcomers to SuperPET or mFORTRAN will find an explanation of the terms used above in the Gazette, Vol.I, p. 256.)

A PASCAL COMPILER FOR SUPERPET
by Associate Editor
Robert Davis

[A few months back, we published a note about the ability of the ZOOM compiler on the C-64 to handle the Pascal ASCII files generated by SuperPET. Since then, Bob Davis has been trying a variant of ZOOM (KMMM Pascal), which runs on the 6502 side of SPET. Ed.]

KMMM Pascal, Level [Version] IV.6E, consists of an editor, a compiler, and a translator, plus some demonstration programs--all on a disk which may be copied. A security key (dongle) must be attached to the No. 1 cassette port to use the compiler--a two-pass program which creates in memory a P-code file from either an ASCII or a PET ASCII source file. The translator creates a machine-language program from the P-code. The machine-language program consists of a run-time package of 6K of support routines plus the machine code created from the source file. You do not have to use the dongle when you run the machine code. The run-time package is copyrighted, but you can make unlimited copies if the copyright display isn't removed and if KMMM Pascal is given credit in any documentation. ZOOM Pascal is essentially level III of KMMM Pascal, but ZOOM is available only on the C-64.

Relative to standard Pascal, you face some restrictions and changes: ARRAY is limited to two dimensions; INDICES are limited to integer sub-range. Enumerated types (e.g., days of the week) as an index are a major strength of Pascal; the index limitation is a potentially serious flaw. An ARRAY of ARRAYS is likewise not allowed; file names are STRINGS instead of PACKED ARRAYS of CHARS. You can address IEEE devices; GOTO is not implemented. One school says never use GOTO; another says there're very legitimate uses for GOTO at times (Peter Grogono). I have some fairly powerful programs which use GOTO legitimately; it would be awkward to rewrite them. PACKED is treated as a comment.

Standard Pascal handles strings as PACKED ARRAYS of CHARS; KMMM employs string functions instead. You find significant differences in programming approach; existing programs must be revised. SET is not implemented--for some, a serious flaw, for the concept is unique to Pascal, at least relative to BASIC and FORTRAN, and often is useful. TYPE - FILE is not allowed. VAR - FILE is limited to TEXT or to component type CHAR or RECORD. WITH is not implemented--which means a bunch more typing and a cluttered program when you deal with RECORDS, but it's not a true limitation. All of the UCSD STRING handling functions and procedures are available except INSERT. DISPOSE is not implemented, which might be a problem with lengthy linked lists (I'm a bit out of depth here). Neither PACK nor UNPACK are implemented; since both are missing from the Waterloo version, this should not be a problem. PAGE isn't implemented; though I'll miss it, it's easy to program around its absence.

You find other limitations: The maximum size of a procedure or function parameter is 80 bytes. A maximum of 132 bytes or so would be nice; then you can send a full line to a dot-matrix printer in compressed mode or print full width on 14-inch paper. A structured type procedure or function parameter, e.g., an ARRAY, must be declared as a variable parameter. This isn't considered good practice, for you risk side effects; you must carefully check any existing, well-written program to live with this limitation. The fields of a RECORD type may not exceed a total of 255 bytes; this could be a problem, but such a limit isn't unusual on many data base systems. The fields of a RECORD type may not be structured. This could be a problem.

After I renamed them in caps so the file names could be read in 6502, I compiled and ran the Waterloo tutorial examples without modification. Results: PEX1-18 and PEX23 ran with no errors. In PEX19, the VAR matrix had to be changed to a two-dimensional array of integers; the program then ran without a problem. PEX20 needed such extensive revision I didn't bother. PEX21 and 22 must be rewritten in terms of STRINGS instead of PACKED ARRAYS. Now, of course, we come to the good part--speed of execution. First, I tried some loops (time is in seconds):

Loop Form:	Variables:	KMMM	Waterloo
for i := 1 to 1000 do	integer	0.73	21.3
v := v + 1;	real	1.90	28.6
repeat v := v + 1	integer	0.36	35.4
until v = 1000;	real	1.90	48.4
while v < 1000 do	integer	0.48	35.0
v := v + 1;	real	2.15	48.0

The Sieve of Eratosthenes for the primes to 1000 requires 5.6 seconds in KMMM Pascal, 27.8 seconds in Commodore BASIC, and just over four minutes in Waterloo's interpreter. On the other hand, a program which solves for three simultaneous equations by Cramer's rule takes 33.4 seconds in KMMM Pascal and only 44.6 seconds in the Waterloo interpreter in SuperPET. The same equations, by Gaussian reduction, take 33.0 and 42.5 seconds. The bulk of the time in all cases is for entry of the required twelve real numbers.

We also face some known problems: READ(LN) and WRITE(LN) when used with files are limited to TEXT or FILE OF CHAR. GET and PUT are limited to FILE OF RECORD. TRUNC and ROUND return a value off by one for negative arguments. I presume that the author of KMMM is at work on this. When I tried a somewhat longer and more complex program to solve simultaneous equations by Gauss-Jordan eliminations, I received a spurious and fatal error message from KMMM. It seems it can't pass a Boolean variale as a procedure parameter (I ran the same program in Waterloo's interpreter without problems). I also discovered that error checking isn't complete; e.g., you get no error message when a number is outside of its declared subrange. And, though it's a minor point, there are no default field width formats for output of various types, as with Waterloo. Comments must be enclosed in

(* and *); underlines must be eliminated (they're interpreted as back-arrows, or carriage returns). The photocopied, note-book style manual is rather brief, with few examples. Extensive familiarity with UCSD Pascal is assumed.

KMMM Pascal definitely has advantage over Waterloo microPASCAL for string handling. It crunches numbers rapidly and should sort arrays quickly. For those who have a true need or a very great desire for a compiler for their Waterloo Pascal programs, KMMM Pascal will work. In view of the many limitations and restrictions, however, it's difficult to recommend KMMM Pascal for compiling Waterloo programs--except that it's the only game in town (pending what may come of the OS-9 operating system).

KMMM Pascal is produced by Wilserv Industries, Box 456, Bellmawr, N.J. 08031, (609) 227-9696, and sells for a suggested $99. Wilserv appears to be a one-man operation--namely, Willi Kusche. I bought my copy from A.B. Computers, 252 Bethlehem Pike, Colmar, PA 18915 (215)822-7727, at just $100, including shipping and handling. Wilserv does not have an 8050 disk drive and must go elsewhere to copy 8050 disks. If that is your format, be prepared for delays. When I sent some source code to Willi (by cassette tape!), he provided very prompt help in determining the cause of the bug in the Gauss-Jordan program. If you subscribe to the "User Library" for $5.50, you are entitled to the latest verion of KMMM Pasal if that was not the one you received. I recently received notice that Level IV.7B will shortly be available; as announced, "this version allows an array to be declared as part of a record," and "there are numerous other minor improvements."

MAPPING EUROPE ON FOOT or Reinventing Radio

After we threw away countless hours learning 6809 assembly language and ran test after test to define how SuperPET works, we learned (sob!) about an easy way. In short, we mapped Europe on foot and then invented radio, only to find both had already been done. We checked the references published last issue (I, 285) and found we didn't have The Commodore SuperPET Computer - Machine and Assembly Language for the Motorola 6809, by D. D. Cowan; it is available from WATCOM Publications, 415 Phillip St., Waterloo, Ontario, Canada N2L 3X2, for $10 (U.S. in U.S., Can. in Canada) plus postage and handling.

All the things you wanted to know about 6809 assembly language and how it's used on SuperPET (well, almost) you'll find in the spiral-bound book, 8.5x11 inches, well-reproduced, full of examples and exercises, and written pretty clearly. If you are learning or want to learn 6809 assembly language on SuperPET, get it. It seems entirely suitable for schools; it starts with fundamentals and builds from the simple to the complex, chapter by chapter--of which there are 20.

The book does have some flaws, but not many. We wish the chapter on macros were a little clearer, that on interrupts longer and more specific, and that on interfacing to the parallel user port a bit easier to follow. But, on the whole, it's well worth the price; we strongly recommend it, especially to those of you who'd like to learn assembly language. With this guide, it's almost painless.

MUMPS ON THE SUPERPET

by Dan Jeffers
Quality Data Services
2847 Waialae Avenue
Honolulu, Hawaii 96826

If you are tired of having to remember all the little differences between the different dialects of BASIC (or COBOL or FORTRAN or PASCAL or any other language), or if you find you are constantly getting bogged down with the details of file I/O and managing large, indexed databases, then I would strongly recommend that you take some time to check out the MUMPS programming language.

MUMPS is an acronym for (M)assachussets General Hospital (M)ulti-(P)rogramming (S)ystem, where it was first developed in 1966 to simplify the interfacing of a wide variety of hospital instruments. From Mass Gen, it quickly found its way out into the business community, where today the more well-known applications include COSTAR (a medical records-keeping system) and the Veterans Administration's File Manager package (a generalized relational database system). Although it was originally associated with the DEC PDP product line, it has since been implemented on a wide variety of systems, including Apple (II and MacIntosh), Burroughs, Data General, Harris, Heathkit, IBM PC (and compatibles), Prime, Tandem, and many others. MUMPS has been used to develop large-scale systems for a variety of industries, including banking, insurance, and medical. There are a number of highly customized applications for auto junkyard inventory control, television network management, etc., and several artificial intellgigence (AI) applications. A very active MUMPS Users Group (MUG) has an extensive following in Europe and Japan as well as in the U.S.; it sponsors an annual conference and periodic regional training seminars, publishes a quarterly digest, and acts as a clearinghouse for an abundance of MUMPS reference material and public domain application systems. An associated MUMPS Development Committee (MDC) provides ongoing review of the language standard, submits proposals for updating the standard to ANSI (the American National Standards Institute), and evaluates how well vendor implementations of MUMPS conform to the standard.

MUMPS, in its traditional form, is an interpreted, interactive language with a number of features that help the programmer to be productive. Once MUMPS is mastered, a programmer is able to concentrate on solving applications problems, not on dealing with the peculiarities of a language or file system. More importantly, however, MUMPS programs written in one vendor's MUMPS easily port to and run on another vendor's MUMPS, often with absolutely no translation or revision of any kind. While the Standard allows implementation-specific capabilities, it defines precisely how such capabilities should be implemented.

The feature that most sets MUMPS apart from other languages is its method of storing information on a disk. Rather than forcing the programmer to deal with conventional files, MUMPS employs hierarchical disk-based data structures called "globals", which are accessed exactly the same way as local memory arrays. (They are called "globals" because on multi-user systems, these data structures are available to many simultaneous users.) Globals are referenced by prefixing an up-arrow character (^) before the Global name, which optionally may be followed by one or more subscripts, either string or numeric. The command at left, for example, causes "JOHN SMITH" to be written to disk (it is usually stored in a memory disk buffer and physically written to disk at a later time). This structure might be appropriate for an accounting system where account # 1234 is owned by someone named JOHN SMITH. This data can be read later into the local variable NAME with the syntax at left.

```
SET ^ACCNTS(1234,"NAME")="JOHN SMITH"
```

```
SET NAME=^ACCNTS(1234,"NAME")
```

The programmer is relieved of any responsibility for knowing or understanding how MUMPS actually stores this data on the disk, as this is completely handled by MUMPS. There are no opens, closes, file numbers, index files, or other anomalies to deal with. All globals are open all the time; there is no practical limit to the number of globals or subscripting levels which may be used. All disk space is dynamically allocated and deallocated as required. Any global may grow to fill all available disk space. When the data is no longer needed, it may be deleted from the disk

```
KILL ^ACCNTS(1234,"NAME")
        or
KILL ^ACCNTS(1234)
```

with either of the commands at left. The second ex- example will delete not only the "NAME" subscript but any other subscripts which may have been stored at the same subscript level as "NAME" under a first subscript of "1234".

One of the functions available in MUMPS allows a program to sequentially access all defined subscripts in any subscript level. This function, known as $NEXT, is used as follows: If our ^ACCNTS global is defined as at left, then the program below causes the numbers 1038 and 1234 to be printed on separate lines (the ! at the end of the WRITE command ouputs a carriage return plus a linefeed).

```
SET ^ACCNTS(1038,"NAME")="JACK WILSON"
SET ^ACCNTS(1234,"NAME")="JOHN SMITH"
```

```
       SET ACCTNUM=0
LOOP   SET ACCTNUM=$NEXT(^ACCNTS(ACCTNUM)) IF ACCTNUM<0 QUIT
       WRITE ACCTNUM,!
       GOTO LOOP
```

Besides globals, many other features of MUMPS make it easy to deal with:

1. All arrays, whether global or local, are totally dynamic and sparse. Totally dynamic means that you need no data definition or declaration statements; totally sparse means that you can define subscript 100 of an array without allocating storage for 99 other elements. In fact, you can define subscripts 100, 399, and 721 and only use up disk or memory space for three data elements. MUMPS will, in addition, keep the array sorted so that you can access the elements sequentially in sorted order (using the $NEXT function as described above).

2. There are only two data types: string and numeric. All data are stored as strings, and are converted internally to numeric as needed, using a straightforward rule. To demonstrate this rule: the string "2.45ABC" converts to 2.45, whereas the string "XYZ3" converts to 0 (zero). In effect, MUMPS simply strips trailing non-numerics from any pure number which leads a string. If there is no pure number leading the string, MUMPS converts the string to zero.

3. MUMPS includes an extremely comprehensive set of functions and operators to manipulate strings. They are too numerous for this article; included are all the common capabilities, such as concatenation, string comparisons, extremely sophisticated pattern-matching, sub-string and sub-piece functions, sub-string "finding" (sometimes call the 'index' function in other languages), and many others.

This truly unique language is available on the SuperPET through Eclectic Systems Corp., 16260 Midway Rd., Addison, Texas 75001, with a list price of $300. This version, known as CCSM (for Comp-Consultants Standard MUMPS, for the software house which developed it), is a full blown standard MUMPS, which I used heavily in 1982 for developing several small application systems. Although this SuperPET version is quite adequate for learning and developing MUMPS systems, I would not recommend it for a production system, as its reliance on Commodore's DOS severely limits the amount of available disk space (globals are contained in a single DOS relative file, which is limited by the DOS to a little over 167K on any one diskette). It is extremely slow on getting global data onto and off of the disk, compared to most other good micro-based MUMPS systems.

The CCSM version is quite adequate for development or learning the language. It contains its own built-in screen editor, tailored for working on MUMPs programs and for developing programs to be run on other MUMPS systems. If you'd like to learn more about MUMPS, I suggest that you contact the MUMPS Users Group (MUG), 4321 Hartwick Road, #510, College Park, MD 20740, (301)779-6555 for more information. Ask for their order form for MUMPS publications.

*   *   *

We sent a draft of the review above to Jerry W. Carroll of 21735 Ybarra Road, Woodland Hills, CA 91364, who recently bought MUMPS. He comments: "Since Dan was using MUMPS in 1982, he may not have the new revision. The price is now $399 but includes the MUMPS development committee Type A extensions included in the 1983 language revisions. (Some of them are shown at left.) Both the string subscripts and $ORDER function are great; $ORDER operates in the same manner with strings that the $NEXT function does with numerical subscripts. A really nice extension to the language.

1. String subscripts and $ORDER function.
2. Range checking on pattern match.
3. A number of function enhancements.
4. Read length control.

"The 8050 disk drive is some help with relative files in that two 720-block global files may be set up on disk. As far as I have been able to determine, only one global file may be opened at a time, however. This slows down access to data which occupies more than one global file, since you have to close one before you access another. An 8250 or 9060 drive might eliminate that problem; I am not sure that the CCSM version of MUMPS supports these drives....the CCSM manuals do leave a lot to be desired."

MORE ON REFERENCE BOOKS Don Momberg of Greek Brook, N.J., writes that he's read the PASCAL references in our last issue, but that the only book which makes sense to him is PASCAL. An Introduction to Pascal and Structured Design, by Nell Dale and David Orshalick, D.C. Heath (1983). He says it contains standard PASCAL; that you change nothing for SuperPET; that it's highly regarded by high school computer science teachers he knows; and that the New Jersey Institute of Technology now uses it for its undergraduates. Don adds that it's easy to read and follow the book.

Bob Davis, Associate Editor, PASCAL, reports that he's read Problem Solving and Structured Programming in BASIC, by Elliot B. Koffman and Frank L. Friedman, Addison-Wesley (1979). He adds that it is one of a series, of the same format as other books by the authors, and instructive to see how the same "experts" approach the same problem in three different languages. Three versions of BASIC are emphasized: Dartmouth, BASIC-PLUS, and ANS Minimal; the flow charts are a bit dated. Well, now; that makes eleven.

Eleven what? Reports on books useful in all SuperPET languages/facilities. From the vast sound of silence, the rest of you read the Gazette and the comics. Of all our teachers, two read books. We guess the rest are too busy with the PTA.

FOR THOSE WHO ENJOY MYSTERY STORIES IN WHICH WE GET TO NO AVAIL Does the telephone always ring as soon as you start to run a long, involved program? If so, you probably learned, long ago, to stuff a GET in the middle of your loops so you can pause a program and restart it again at your pleasure. If your language has no GET (mPASCAL and mFORTRAN don't), you can write simple routines which will sense a keypress without a RETURN (we demonstrate some later), whether to pause a program, to pick a choice from menu, or to control execution. In exploring the mystery we next describe, we picked up some knowledge and techniques useful in all languages.

Whatever the language, however, GETs sometimes don't GET; at times, you can hold down a key with both feet and never sense the keypress. Hmmmm. Why? (For those unfamiliar with GET: SPET allows you to sense the ordinal [ASCII code number] of any key as a numeric variable, unlike Microsoft BASIC, which GETs only string values, except for the number keys.) You can, of course, GET a key as a string.

Let's start at the beginning. In the loop below, we can pause our program at any time with OFF. Since you needn't clear the keyboard buffer or zero out variable "ordinal%", we usually use this method in all loops of long programs--a touch of OFF pauses a program; another touch of OFF restarts it. Well and good--until you vary the method to sense any key but OFF. Woe, woe. We revised the program to pause on "p" and to continue on "c", (see the second program below) and ran into two problems, one minor, one major. The minor problem: Any key but OFF, if held down, not only fills the buffer, but also takes priority (while it dumps from the keyboard buffer) over the next key you press. In sum, you must clear the buffer of one character before you employ another. The program below does that. But!--we run into another and more serious problem. Take a look at line 170, below. Will the program proceed while the "c" key (ordinal 99) is held down? Yes or no?

```
10 loop
20   ...rest of loop
30   get ordinal%
40   if ordinal%=255 ! OFF=255
50     get ordinal% : if ordinal%<>255 then 50
60   endif
70 endloop
```

If you said "yes," we're sorry (sob!) to say you're wrong though eminently logical. If you watch the printout of "ordinal%", you'll often see zero, even if the "c" key is held down by an anchor and both feet.... Why? How does a zero ever return if the "c" key is depressed? While you ponder that mystery, note that lines 130-180 will pause any loop with "p" and continue it with "c" no matter how long either key is held down.

```
110 ! insane:bd
120 for i%=1 to 1000
130   get ordinal%
140   if ordinal%=112      ! 'p' key
150     get ordinal% : if ordinal%<>99 then 150
160   elseif ordinal%=99   ! 'c' key
170     get ordinal% : if ordinal%=99 then 170
180   endif
190   for k=1 to 150       ! Vary this delay to
200   next k               ! see what happens
210   print ordinal%;
220 next i%
```

If you're curious, play with the time delay in 190-200, or take it out. Peek the keyboard buffer (decimal 304 to 343); see if you can figure out what happens to the missing "c". If you're puzzled, write a little program which reports time in seconds, PEEKs the jiffies in SuperPET (you'll find 'em at $163), and prints the ordinals of the characters you GET. Run it. How many zeros come back instead of characters when you hold a key down? Stuff in a time delay. Do things change?

If you figure out what goes on, then write a routine which will pause a program as long as a key is held down--and always pauses it when a key is depressed, no matter how briefly. If you can't penetrate the puzzle, read on whilst we explore the keyboard, the PIA and the keyboard buffer. All may become clear (well, mostly). Best of all, we learn to make the keyboard dance to our tune in all of the languages.

HOW TO TRICK THE KEYBOARD BUFFER AND USE ITS POINTERS

SuperPET carries two pointers to the keyboard buffer, at decimal 300 and 302. We knew the first pointed to the location in the buffer (at decimal 304-343) of the next character to be sent to screen, but hadn't figured out what the second pointer (at 302) was for. Whilst puzzling over the mystery described in "For Those Who Enjoy Mystery Stories," this issue, we finally found out--and it's handy to know (each pointer is in two bytes; at 300-301, and at 302-303 decimal).

The keyboard buffer holds 40 characters (well, 39. See below). How does SuperPET know which is to be printed, and when to stop printing?

Suppose you hold a key down more than half a second or enter a string of characters (the repeat routine repeats a key when it's down for 30/60ths of a second). The first pointer (at 300) marks the position of the first character put into the keyboard buffer; as a key repeats or more characters are entered, the second pointer increments. When the last character is entered, the second pointer (302) points to the next buffer location to be filled. As characters print from the buffer, the first pointer (at 300) increments until it equals the second pointer (at 302). Well, if both pointers point to the same location, one location in the buffer can't be filled--which is why you'll print only 39 characters from the buffer (thanks, Loch Rose). Obviously, when both pointers point to the same address in the buffer, you get no more characters from it until you press another key. If you use this knowledge, you can fool a full buffer into thinking it's empty. And you can, as we'll later see, easily find the last character of many entered into the buffer.

```
100 ! foolit:bd
110 open #12, "keyboard",output
120 for i%=1 to 40
130    print #12, rpt$("X",40)
140    get ordinal%
150    print ordinal%;
160    poke 300,1,48,1,48
170    get ordinal%
180    print ordinal%;
190 next i%
```

In the program at left, we fill the buffer with trash (forty X's) on each pass, print one character of the trash, and then POKE the two buffer pointers to the value of 304, the address of the first location in the buffer (remember that a POKE of 1 in the high byte of an address equals 256; 256+48=304). When we try to GET another character on line 170, the buffer says "Ain't got any," and returns zero. Run the program as is; then comment out line 160 (the POKE). Aha! You print the ordinal of "X"--and when the program ends, you dump all 40 X's from buffer to the screen. Stuff the POKE back in and the trash disappears--in all languages.

Those with sharp eyes will note that the high byte of each pointer will always hold a "1"--for the decimal 256 which, added to the low byte, forms the buffer address. So you can, as Loch Rose suggests, fool the buffer by "poke 301, peek (303)". This stuffs the same low byte into each pointer; we know the high pointer bytes always are "1". Or you can "poke 301,48, poke 303,48", which sets both pointers to the start of the keyboard buffer at 304 (256+48=304).

You may, of course, clear the whole buffer in some languages by opening the keyboard as a file and printing 40 nulls [print #5, rpt$(chr$(0),40) fills the buffer with nulls in mBASIC, for example]. This won't work completely in mFORTRAN, however. Part of the buffer is filled with nulls; part remains full of previous characters; in addition, you must use format control and print the nulls with an "a1" format, or you get a space (ASCII 32) after every null. Thus the POKEs discussed above are particularly useful in mFORTRAN, as we'll later see.

Equally useful is the knowledge that you'll find the very last character entered at the location pointed to by the second pointer (at 302) minus one. If that pointer always points to the next location to be filled, then "high pointer-1" must show the address of the last character we stuffed into the buffer (except when the pointer "wraps around;" see below). We'll later put this knowledge to work. Before we do, let's look at the PIA (Peripheral Interface Adapter) at address $E810-$E813, which actually senses all keypresses in SuperPET. We will be able to exploit that device, too--in all languages.

* * *

Pointer Wrap-Around Associate Editor Stanley Brockman reminded us that both pointers (at 300 and 302) must "wrap." The Keyboard Buffer can hold only forty bytes; when either of the pointers reaches location 343--the end of the buffer--it wraps to location 304. Thus, if you PEEK the pointer at 302, and it holds the address of 304 (start of buffer), the last character entered in the buffer is at location 343--and not at location 304 minus one. If you want to sense accurately the last character entered, you must allow for this "wrap." At other times, when you sense a keypress, you may disregard "wrap." After all, if you hold down a key, and the high pointer (at 302-303) has just "wrapped" to index the start of buffer at 304, you get the answer to what key is down 1/60th of a second later, at next IRQ. Then the high pointer increments to 305, and the last character you enter will indeed be found at 305 minus one--at location 304 in the buffer. We sense keypresses in later examples in this issue without regard for "wrap."

A GLANCE AT THE PIA
Another Approach

The PIA1 (1st Peripheral Interface Adapter) senses keypresses in SPET during IRQ interrupts. Software in ROM uses PIA1 to scan the keyboard sequentially, a row of eight keys at a time, for ten different rows and a total of eighty scans. The first byte in PIA1 ($E810) controls which row of keys is looked at. In the table below, for example, we find that if $E810 contains $F0, the "5" key on the main keyboard, if down, will return $FD in PIA1 at $E812. Any of nine other keys can also return $FD if down, but only if the value in $E810 is changed. If no key is down, $FF returns in $E812. As we'll see, that knowledge is golden in any SuperPET language.

The ROM routine which drives the PIA scan always starts with a value of $F0 in the PIA at $E810, and then increments that value through $F9 as it scans more rows of keys. The last row of keys scanned is shown in the right-hand column of the table below, under $F9. Since the $F9 scan is the last look at the keyboard, the returned value in the PIA (if any) for any key in that row remains in the PIA, at $E812, until the next interrupt. You can therefore peek the PIA at $E812 and identify any key in that last row, if it was pressed. In short, if you use the colon, 9, 6, 3, or left-arrow keys for program control, you can sense which key was pressed--and if it has been released. We do it in a simple way in mFORTRAN at left, below, and give that language a way to GET a keypress without a RETURN in a routine which runs quickly. We use the LEFT ARROW key; when pressed, it pauses the program until the key is released. It then looks for another press of LEFT ARROW before it will resume. Obviously, you can modify the routine to force mFORTRAN to do anything you want done when LEFT ARROW is touched.

```
do i=1, 40              * The DO loop is purely for demonstration.
  print, "loop"
  call get                 * A call to subroutine GET checks the PIA to see if
enddo                      * a specified key has been pressed.
end
                           * The negative peek (-6126) is at location $E812.
subroutine get

  ipia=peek1(-6126)        * We peek the PIA at $E812.
  if (ipia=254) then       * If the key value is $FE (254), LEFT ARROW is down.
    loop
      ipia=peek1(-6126)    * Wait until $FF (255) says no key is down.
    until ipia=255
    loop
      ipia=peek1(-6126)    * Program pauses until LEFT ARROW is pressed again.
    until ipia=254
    k=poke2(300,304)       * We tell the keyboard buffer it is empty. See article
    k=poke2(302,304)       * this issue on fooling the keyboard buffer. If this
  endif                    * is not done, the LEFT ARROW key will print to the
end                        * screen.
```

Table of PIA Key Values at $E812 vs. PIA Values at $E810
This table shows which keys are scanned for each of ten values in the PIA at $E810, and the return value for that key as found in $E812.
['K' stands for 'Keypad']

Value in		Keyboard Row Values in PIA at $E810									
$E812		$F0	$F1	$F2	$F3	$F4	$F5	$F6	$F7	$F8	$F9
$7F	Key:	Vacant	K9	K5	K6	DEL	K4	K3	K2	K1	Vacant
$BF	Key:	Vacant	Vacant	;	@	p	[	R.SHFT	Repeat	/	Vacant
$DF	Key:	Cur Rt.	^	k	l	i	o	Vacant	Vacant	Vacant	:
$EF	Key:	K8	K7	]	RET	\	Cur Down	K.	K0	HOME	STOP
$F7	Key:	-	0	h	j	y	u	.	,	m	9
$FB	Key:	8	7	f	g	r	t	b	n	Vacant	6
$FD	Key:	5	4	s	d	w	1	c	v	x	3
$FE	Key:	2	1	ESC	a	TAB	q	L.SHFT	2	OFF	_

There are 71 keys to which the PIA can respond (not counting shifted keys or the SHIFT LOCK key); the matrix above provides room for 80 values, of which nine are vacant. The abbreviation 'K' stands for 'KEYPAD'. The normal value seen in the PIA at $E810, outside of interrupts, is $F9; we restore it (though it may not be neccesary) when we use $E810 to sense a keypress directly. The Interrupt Flag should be set (SEI) before varying $E810, and restored (CLI) at the end of the assembly-language segment which varies $E810, else the normal interrupt routine will go through the keysense routine and change the value in $E810. Note that the right and left SHIFT keys return two different values. Sensing SHIFTed keys requires a complex program, and is not advised unless you understand the PIA thoroughly.

In assembly language, you may sense specific keys during a user interrupt routines while the Interrupt Flag is set and normal system key-sensing routines are not available. The short assembly language segment below shows one way to sense a specific key while the Interrupt Flag is set. The routine will not execute until OFF is pressed and released ($FF returned in $E812 says "no key is down").

```
printf_ equu $b0b7
sei                   ; Set the interrupt flag.
ldd  #message
jsr  printf_          ; Tell user to press the OFF key to proceed.
lda  #$f8
sta  $e810            ; Store in $E810 proper value for OFF key ($F8).
loop
  ldb $e812           ; Load PIA key value returned.
  cmpb #$fe           ; Compare with value of OFF key.
until eq
loop
  ldb $e812           ; Continue sensing $e812 until NO KEY DOWN is returned.
  cmpb #$ff
until eq
;Insert routine to execute when OFF is pressed; we substitute a simple message.
ldd #doit
jsr printf_
ldb #$f9              ; This may not be necessary, but let's be tidy.
stb $e810             ; Restore $E810 to normal value.
cli                   ; And clear the Interrupt Flag.
swi                   ; This program runs in the monitor only.

message fcc "Press OFF to Execute%n"
        fcb 0
doit    fcc "We execute user routine%n"
        fcb 0
        end
```

IN WHICH WE FIND THE MISSING "C" The "c", of course, isn't the head of MI-6 of British intelligence, but the missing character we lost a few pages back. If you explored the problem, you may have noted the essential clue: in tight, fast loops you always try to GET more characters from the keyboard buffer than there are interrupts to sense them. If, for example, your program prints out 110 GOT characters, while only 80 interrupts occurred, then you are bound to GET some nulls from the keyboard buffer. The problem is made worse because a key won't repeat unless held down for more than one-half a second, and because four interrupts must occur for each "repeat" of a key into the keyboard buffer (the system routine causes this delay so the cursor won't move like lightning).

We must distinguish between key-sensing (done at the PIA during interrupt) and key-getting (done from the keyboard buffer). If a loop executes quickly, there may be too few interrupts to GET a key; if so, no key value is put into the buffer; any GET from the buffer returns a null. That is why "c" was missing, and why a time delay in GET loops reduces the number of "no key down" returns.

If you hold a key down and clear the buffer afterward, you can GET even in the languages without an intrinsic GET. We demonstrate below with a simple program in mFORTRAN, which employs all the techniques we've discussed so far, except the PIA. It pauses any mFORTRAN program on "p" and continues it on "c", no matter how long either key is held down. It can easily be modified to handle menus or to control program execution by GETting a character from the keyboard without a RETURN; the technique should be easy to adapt to any language which does not possess an intrinsic GET function.

```
do i=1, 40
  print, "loop"
  call get
enddo
end

subroutine get

  ipointer=peek2(302)-1          * Get pointer to last character entered.
  ioff=peek1(ipointer)           * Peek the character at that address in buffer.
  if (ioff=112) then             * Is it "p"? (ASCII 112='p'.) If so,
    open(10, file="keyboard")    * Open the buffer, and
    write(10,"a1") rpt(char(0),40) * fill as much as possible with null. If you
    close(10)                    * leave out this step, mFORTRAN will crash if
    loop                         * "p" or "c" are held down long enough.
      ipointer=peek2(302)-1      * Then find the last character pointer again.
      igo=peek1(ipointer)        * And wait for a "c" (ASCII 99).
    until igo=99
    k=poke2(300,304)             * Tell the keyboard buffer that it is empty.
    m=poke2(302,304)             * (You can substitute k=poke1(301,48) and
  endif                          * k=poke1(303,48) to stuff the value of 304
end                              * into each pointer. High byte of both is 1.
```

Though the program seems involved, it runs very quickly. If you've pressed no key, the subroutine runs with two PEEKs and then returns. If a key has been held down, the reaction time you need to press a new key is far slower than the code. The method, of course, is system-dependent and won't work if you transfer the code to another computer running mFORTRAN. Anybody know how to do the same thing in either mFORTRAN or mPASCAL in a system-independent way?

```
100 ! pause:bd
120 for i%=1 to 100
125   get ordinal%
130   print "pauses",
150   if ordinal%=112  ! p key
155     for j=1 to 7   ! pauses
160       get ordinal% : if ordinal%=112 then j=1
165     next j
170   endif
195 next i%
200 stop
```

If you merely want to pause a program while a key is down, and resume again when the key is released, the method at left works nicely. The loop delay is sufficient to bypass null GETs. Neither PASCALiers nor mFORTRANners can use it, however, since you may not reset "i" in either language. In those languages, you are therefore forced to either PEEK the PIA, as we do in a previous article of this issue, or to use a variable you can <u>reset</u> within a loop. The PIA technique is by far the simpler.

TO LAUNDER THE APL WORKSPACE

From Jakob Bennema in Wageningen, the Netherlands, we received some excellent APL printouts which we know emerged from the ISPUG articles on printing the APL character set (I, 196 ff), since Jakob told us so. When we see such results from the hard work of our authors, we hope they feel repaid, as we do, for our time and trouble.

Jakob apparently ran out of workspace--a not uncommon problem--and had to cudgel out room for a few more bytes. Though he)ERASEd both functions and variables, he checked free memory with quad WA and symbols with)SYMBOLS--only to find that while his free memory had increased, the symbols list was as long

(Function on next page)
Page breaks! $#%@#!

as ever. You can follow what he did in our example at left, where we delete some functions and some variables from workspace and check free memory and the number of symbols before and after.

```
CLEAR WS
      )LOAD APLCHARS2(AWS
SAVED  84/01/22  14:45:42
      ⎕WA
17355
      )SYMBOLS
200 : 69 IN USE
      )ERASE CENTER             DELETE FUNCTIONS
      )ERASE PRINT
      )ERASE OPEN_PTR
      )ERASE APLCHARS           DELETE VARIABLES
      )ERASE APLNAMES
      ⎕WA
20851
      )SYMBOLS
200 : 69 IN USE
      )WSID TEST
      )SAVE

      )CLEAR
      )LOAD TEST
      )SYMBOLS
200 : 69 IN USE
      ⎕WA
20851
      )CLEAR
      )COPY TEST
      ⎕WA
21299
      )SYMBOLS
200 : 47 IN USE
```

Note that while free memory increases, the number of symbols doesn't go down, even though some have been deleted. So, we rename the WS and save it to disk as file TEST, as Jakob did. Then we)LOAD the workspace and again check quad WA and SYMBOLS. We find no change in either value. Gee, can't we free up a few more bytes for all the stuff we deleted?

As Jakob points out, we indeed can. If we)COPY file TEST into a clear WS, and again check WA and SYMBOLS, lo!, we pick up 448 bytes, and the number of symbols drops from 69 to 47. Can we now save this laundered workspace? Indeed we can, if we give it a name with)WSID NEWNAME and)SAVE it.

Jakob comments that the SYMBOL table defaults to 200; each symbol requires six bytes; you lose 1200 bytes to SYMBOLS unless you set the number needed, which may be good practice in learning how large a symbol table you really must have.

V1.1 APL obviously does not clean up the symbol table when functions and varia-les are deleted. You can, however, "launder" a workspace if you follow Jakob's procedure. You'll face one problem, though: automatic presentation of general information and of menus will disappear from a WS so COPY'd and laundered. You must repair the damage.

T H E   A P L   E X P R E S S   by   R E G   B E C K

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

It was with some apprehension that I accepted the job of APL Associate Editor. Steve Zeller has moved on to the IBM PC. His act will be hard to follow!

First, some info on your new Associate Editor-- I teach senior high school physics, math, and computing. APL is used as an alternate programming language to BASIC and for enrichment work in our computing science course. I write APL programs for classroom use and for fun, am a committed SuperPETter, am not an APL programmer of the kind who can save the world in one line of code, and can't promise to solve every reader's problem.

I will, however, try to write an interesting column with hints and kinks on the APL system, useful programs, explanations of some of the fundamentals of the language, and ideas for teachers. Reader participation is, of course, welcome.

Send me your ideas, problems (and if possible, your solutions) and your favorite functions. As with Steve's column, all work will be done in version 1.1 Waterloo microAPL.

To begin, in this first column, let's look at the problem of input and ouput. Many beginners find the Waterloo manual less than adequate on this subject. The commands and functions are defined in the manual but require clarification and examples. In this column, I'll examine workspace and system commands (files will be covered in the next).

The commands we examine are defined in chapter 2, page 42 of the Waterloo APL manual. I'll restrict examples to device #8, the normal setting for Commodore dual drives. The language may be loaded from either drive. To load from drive 0, enter either disk/0.a or disk.a <RETURN> from main menu. The loader at menu knows the "a" is an abbreviation for "APL." All commands must be followed by a press of the RETURN key; I won't continue to show the RETURN in this column.

First, some definitions of terminology used:

term	definition
library	disk directory of saved files
program file	a disk file containing program material
workspace	APL program containing functions and variables
active workspace	workspace currently in use in R/W memory
object	any single function or variable in a workspace
wsid	name of a workspace (WorkSpace IDentification)

1. Displaying libraries: This doesn't require much comment. Use)LIB for drive 0 and)LIB DISK/1 for drive 1.

2. Workspace names: Display the name of the active workspace with)WSID. You may rename the active workspace in this manner:)WSID EXAMPLE, "example" being its new name.

3. Loading in a program file from disk: The named program file will load in and will become the active workspace. The previous workspace will be gone. For drive 0 use:)LOAD wsid, where wsid is the exact name that appears on the library listing of drive 0. Example:)LOAD SORT loads in a PRG file named SORT from drive 0. The name of the active workspace is changed to SORT.)LOAD DISK/1.SORT loads SORT from drive 1. The active workspace is, however, named DISK/1.SORT, and not SORT, as you expect. This pops up again when you use the)SAVE command or some of the QUAD functions in file I/O. No benefit derives from this syntax as far as I can see.

4. Copying objects from a saved file into active workspace:)COPY and)PCOPY are the two forms of this command. The name of the active workspace does not change when you use these commands. Objects in the active workspace which have the same names as incoming objects are replaced when you use)COPY.)PCOPY will not load objects which have the same name as objects already in the active workspace. To copy an entire workspace (except for the limitations above), use)COPY wsid or)PCOPY wsid, where wsid is the name of the disk PRG file.)COPY SDUMP, for example, copies every object in the PRG file SDUMP from drive 0 into the active workspace. For drive 1, use)COPY DISK/1.SDUMP. When you wish to copy in

certain named objects only, follow this statement with a string of names separated by blanks:)PCOPY EXAMPLE DEFINE DESCRIBE A B will copy the objects DEFINE, DESCRIBE, A and B into active workspace (as long as no objects having such names already exist in the active workspace).

This is a slow procedure as the system searches sequentially through all objects in the PRG file until it finds the objects named.

5. <u>Saving workspaces as program files:</u>)SAVE will save the active workspace, under its present name, as a PRG file on drive 0. The obvious)SAVE DISK/1 to save the WS to drive 1 fails; the name of the active WS is changed to DISK/1! In order to save to drive 1, you must use one of two methods: 1) prefix DISK/1 to the name of the active workspace, as in)SAVE DISK/1.EXAMPLE (note that the name of the active workspace becomes DISK/1.EXAMPLE), or 2) change the name of the active workspace, with:)WSID DISK/1.EXAMPLE, and then)SAVE the workspace. It will file to disk on drive 1.

If you wish to save the active workspace under a new name, use)SAVE followed by the new name. This operation will abort if a PRG file with this name already exists on disk. Of course, when we look at the WSID after this operation, we find that it has been changed to the new name. If, for example, the current name of the active workspace is EXAMPLE and a)SAVE EX is successful, the name of the active workspace becomes EX after the SAVE.

6. <u>Scratching PRG files from disk:</u>)DROP wsid scratches the program file named.)DROP EXAMPLE and)DROP DISK/1.EXAMPLE will scratch the PRG file EXAMPLE from disks in drive 0 and drive 1, respectively. This command will not work with any other file types (such as SEQ).

This completes my discussion of the right parenthesis commands.

Disk operations such as formatting a new disk, copying files between drives, backups, etc., can be difficult within the APL operating system. A simple solution to this problem is to have a set of functions on disk which handle these operations. Either copy into workspace specific functions for each operation or use an all-purpose, menu-driven program. Soon after I got my SuperPET, I wrote such a menu-driven program (DOS_SUPPORT). It is found on the ISPUG Starter-Pak and ISPUG APL Character Set disks and is very handy. I'll explore this further in a future column.

Alphabetic sorts are always of interest. In APL, the obvious method is to use the powerful primitives for matrix operations. Character data are entered as rows of a character matrix. The rows may then be sorted according to the alphabetic position of characters in the columns. Simple sorting functions typically are one or two lines long. The following three functions comprise a program to enter, sort, and print a list of student names.

```
        ∇ENTER[⎕]∇
[ 0]     ENTER ;A;I
[ 1]    ⎕TC[5,3]
[ 2]    'ENTER NUMBER OF NAMES TO SORT'
[ 3]    MAT←((N←⎕),25)ρ' '
[ 4]    ⎕IO←I←1
[ 5]    START: →(I=N+1)/FIN
[  6]          'ENTER NAME NUMBER ',⍕I
[  7]          MAT[I;]←A,(25-ρ(A←⍞))ρ' '
[  8]          I←I+1
[  9]          →START
[ 10]   FIN:   SORT
      ∇SORT[⎕]∇
[  0]     SORT               ⍝ WOOPS! REG SENT A SHORT SORT WITHOUT 'MAT←' ON
[  1]   MAT←(MAT[⍋MAT;])  ⍝ LINE 1--IT SENT THE UNSORTED LIST TO PRINTER.
[  2]   MAT                  ⍝ GEE, ANOTHER ONE-LINER BITES THE DUST. SORRY, REG.
      ∇PRINT[⎕]∇
[  0]     PRINT
[  1]   'IEEE4' ⎕CREATE 1
[  2]   MAT ⎕PUT 1
[  3]   ⎕UNTIE 1
```

In line 3 of ENTER, an empty matrix, MAT, is created with N rows and 25 columns. If more than 25 characters are needed, change the number 25 in lines 3 and 7 to the desired number. In line 7 of ENTER, each entered name is put into a row of MAT; the proper number of spaces is added to fit each row into the 25-column matrix. In PRINT, a file is created to an IEEE-488 printer; the matrix is sent to that file. MAT is in the active workspace and may be displayed. When you want to print, simply enter: PRINT <RETURN>.

I brought home a 2031 single drive the other day. It worked fine when treated as drive 0. To load APL from 6809 menu, you must use "disk.a", as noted earlier in this column. The 2031 is read-compatible with dual drives but isn't write-compatible, as the track widths are different. In order to have two drive units in operation simultaneously, the device number of each must be unique. Without modification to the drive hardware, all drives come up as device #8. You may change device numbers temporarily (they will be reset to #8 the next time you turn on the drives) from program, using the memory-write command. This procedure is described (using BASIC) on pages 56 and 57 of Commodore's Disk System User Reference Guide. The memory-write command is sent to the command channel followed by the low byte and then the high byte of the DOS memory address (in decimal form). The data bytes in decimal form are sent next. All drive units (the 4040, 8050, 8052 and hard disks) except for the 2031 use the same DOS memory address.

DOS memory address contents may be read from the drive into the computer using the memory-read command. If we use that command, we can write an APL function which first checks the drive unit and then sends the correct address bytes to whichever drive we use. The example following shows the syntax for such commands in APL.

```
     ∇TO[⎕]∇
[  0]     A TO B ;⎕IO;CHAN;C
[  1]   (CHAN←'IEEE',(⍕A),'+15') ⎕CREATE 1+⎕IO←0
[  2]   ('|+ρ',⎕AV[192 224]) ⎕PUT 1
[  3]   CHAN ⎕TIE 2
[  4]   C←⎕AV⍳⎕GET 2 1
[  5]   ('|+ω',⎕AV[(12+107×C=221),0 2,B+32 64]) ⎕PUT 1
[  6]   ⎕UNTIE 2 1
```

8 TO 9 <RETURN> changes the device number of a unit from #8 to #9.

I arbitrarily picked ROM address $E5C2 because it differs in contents between the 2031 and the 4040 drives. Since the 2031 has a 221 there and the 4040 a 22, you can distinguish between the drives from program.

In line 1, we access the command channel for the M-W and M-R commands, which appear in lines 2 and 5. We must reaccess the channel to input the byte from the drive's ROM. This we do in line 3; in line 4, we read the byte. Line 5 shows how the function selects which address to send. The value of 12+107xC=221 will be 119 if C=221 (a 2031 drive) and 12 if C has any other value (a 4040 drive). This function should also work with other drive units unless we're unlucky enough to get a 221 in $E5C2 from those drives. Try it and let me know what happens.

In future columns I'll try to clarify more I/O commands and functions, cover direct function definition in APL, and touch on simulation. The size and subject matter of the column will depend on the letters, questions, ideas and information I receive from you. Be sure to write! If anyone out there has good memory maps for the 2031, 4040, and 8050 drives, PLEASE send them.

BITS BYTES & BUGS by Gary Ratliff, Sr.

215 Pemberton Drive, Pearl, Mississippi 39208

There is within your assembler another assembler waiting to get out. This is not documented in the Waterloo manual; however its existence is clear to anyone who will carefully examine the reserved word list for the 6809 assembler. This table is found in chapter 9 of the assembler manual on pages 191-193. We invite you to take a few moments to study this table closely.

Yes, here you will find several reserved words which are not for the 6809 micro-chip. The words such as LDAA, LDAB, CLC, DES, DEX, etc. are familiar to those who have seen listings for the earlier Motorola chip, the 6800. Can it be that we have within our familiar 6809 assembler the beginning of a cross-assembler for the 6800?

The answer is almost. Here, I must thank the staff of McGraw Hill for their permission to include the data from Table 3-10 of their "6809 Assembly Language Programming" by Lance Leventhal. It is this table which will allow us to create a set of macros which will give us full suport of the 6800 instruction set. This is because some but not all of the missing 6800 instructions have been converted by the people at Waterloo. It is as if they left out several as an exercise for students.

The easiest way to determine which of these we must convert is to simply enter this table and let the assembler error messages tell us which of these codes are not supported. This is exactly what I did to produce a list of the words which we'll include in macro definitions to complete the 6800 instruction set.

```
*** Error: Undefined operation code: ABA
   1                            aba
*** Error: Undefined operation code: CBA
   2                            cba
   3 0000    1C    FE           clc
   4 0002    1C    EF           cli
   5 0004    1C    FD           clv
   6 0006    32    7F           des
```

Here you will notice that there are only four 6800 instructions which are not already properly converted for us.

These are clearly marked in the list file produced by trying to assemble the complete 6800 set

```
  7 0008    30    1F              dex
  8 000A    32    61              ins
  9 000C    30    01              inx
 10 000E    34    02              psha
 11 0010    34    04              pshb
 12 0012    35    02              pula
 13 0014    35    04              pulb
*** Error: Undefined operation code: SBA
 14                               sba
 15 0016    1A    01              sec
 16 0018    1A    10              sei
 17 001A    1A    02              sev
 18 001C    1F    89              tab
 19 001E    1F    8A              tap
 20 0020    1F    98              tba
 21 0022    1F    A8              tpa
 22 0024    1F    41              tsx
 23 0026    1F    14              txs
*** Error: Undefined operation code: WAI
 24                               wai
 25                               end
```

of "missing instructions."

Also notice that the 6809 assembler has properly translated the missing codes to standard 6809 operations:

For example: the code for "clc" is translated into 1C FE; when this is looked up it is found to mean ANDCC #$FE, which, of course, clears the carry flag in bit 0. The other operations are also correctly translated.

Thus, the only work for us is to supply the macro definitions for these assembler mnemonics: ABA, CBA, SBA and WAI, and our assembler will be ready to do double duty by now being able to act an a 6800 cross assembler.

The data provided in Table 3-10 of the above-cited work will enable us to create the macros very easily. The following set of definitions will let us enter routines from the vast library of available 6800 material and have it run on our SuperPETs.

```
; 6800 macro definitions to allow Waterloo 6809 assembler to use 6800 code
; by Gary L. Ratliff --permission to use Table 3-10 granted by McGraw-Hill
; File all to disk holding your source code as: "6800.asm"

aba macr ; add the 6800 aba instruction
    pshb       34  04
    adda ,s+  AB  EØ
    endm

cba macr ; add the 6800 cba instruction
    pshb       34  04
    cmpa ,s+  A1  EØ
    endm

sba macr ; add the 6800 sba instruction
    pshb       34  04
    suba ,s+  AØ  EØ
    endm

wai macr ; add the 6800 wai instruction
    cwai #$ef   3C    EF
    endm
    end
```

The only remaining problem is how to interface these definitions with our source program.

This we do with the "include" directive. Whenever we want to use these macros, to get full support from the 6800 instruction set, we only have to use the following line in source (assembler) code

```
;include <6800.asm>
```

as the first line in our program and we are all set. As a matter of fact, if you enter the complete list of 6800 instructions, as we did in the example above, you will find that the 6800 instructions will assemble without error if you add the "include" line and have filed the macros above to disk.

There are plenty of programs available for the 6800 chip. A good series on the creation of an operating system was presented in EDN, all of it written for the 6800. Also, the much cited Lewis text, "Software Engineering for Micros," has material for the 6800.

This brief example shows how easy it is to now use 6800 code on SuperPET.

```
;include <6800.asm>

       org $1000
stack  rmb 1
status equ $f000
vari   fcb $5
entry  ldaa vari
loop2  staa save
       lds  #stack
       rora
       bvc  loop2
       ldaa status +1
       rts
save   fcb 0
       end
```

You'll note that in this example we do not use any of the macro definitions we developed. This is to be expected, as the Waterloo assembler uses the macros on disk and translates the 6800 code to 6809 code for us without any problems.

One feature of many 6800 assembler listings isn't supported. Many such 6800 programs show "ldaa" in the form "lda a". This will only produce syntax errors on the 6809 assembler, so use the form "ldaa".

That's it for this time; in the next installment, we'll demonstrate how to write programs which may be assembled on either the 6809 or 6502 processors of SuperPET.

[Ed. Want a printer that lists for $495, prints quietly at 150 cps, turns out very clear copy, and doesn't need an interface (cables directly to the IEEE-488), is mostly compatible with WordPro and PaperClip, and which you can both use and control from the microEDITOR? Read on.]

USER REPORT : HEWLETT-PACKARD THINKJET PRINTER

by Shawne Ross
3029 Keighly Road, Nainamo,
B.C. Canada V9T 2C9

My requirements for a printer were: good print quality, especially when duplicated on a Xerox machine, easy to use, simple to interface, reasonably priced and not too noisy. Much of my word processing is done with the microEDITOR, as I often mix text and program listings in hand-outs for my computer science classes, so I particularly wanted a printer that would respond well to commands given from the microEDITOR, as well as work with WordPro.

The HP ThinkJet comes with one of three interfaces built-in: the HP IB, Hewlett-Packard's version of the IEEE-488 bus, parallel, or serial. I received somewhat conflicting advice on which of the three to choose: one Commodore dealer suggested that the HP IB would not be compatible and that I should get the parallel which would require the purchase of the Centronics Parallel interface at a cost of $200 additional; another Commodore dealer suggested that the only real advantage of the Centronics interface would be that the printer could be transferred to another system if I changed computers; and the Hewlett-Packard dealer could see no reason why the HP IB would not work just fine. Being a little short of cash, I chose the IEEE (HP IB) and simply cabled the printer onto the disk drive with an IEEE cable and so far have found no problems whatsoever with this setup.

Two small changes were required to get myself printing. The printer has a seven-segment switch on the rear panel which is used to set the printer address. It is factory set for an address of 1, common to HP systems, and needs to be set to 4 for the SuperPET if you want to call your printer "ieee-4". The rightmost five switches need to be set at 00100, which represents 4 in binary.

The second essential change is to send a one-time command to the printer, after you turn it on, to generate a linefeed whenever it is sent a carriage return. Otherwise, all of your text will be printed on one line--most disconcerting the first time it happens. The manual very clearly explains how to send this code, and I found in all cases that best method was to send a series of ASCII numbers. Since the linefeed is needed in all modes of operation, including WordPro, I first wrote programs in Pascal and in Basic (6502) to send the command. Later, I saved the character codes as a disk file which can be sent from the mEDITOR, then finally wrote a short assembler program which can be called from main menu to do the trick. This command needs to be sent only once--on power-up.

I found the manual well-written anu easy to understand. It contains all I need to know to access the various modes of the printer, including four different pitches, 6 or 8 lines per inch, perforation skip, bold print, subscript or superscript and underlining. The nicest part is that all of these modes can be invoked in the microEDITOR using the method given in the April/May Gazette, (Volume I), page 220.

To try out the dot-addressable graphics, I designed a small logo of my initials and followed the instructions in the manual to write a Pascal program to produce it on the printer. It is quite easy to do, although a little boring and time-consuming--but another bonus popped up. With two small changes to the Pascal program, I could send the characters to a disk file [declare a variable "graphic" of type "file of char", then add a line: rewrite(output, 'graphic')]. After that, whenever I want to print the logo I can simply say "copy graphic to ieee4" in the mED and it prints, considerably faster, incidentally, that from the Pascal program.

I consider the print quality to be very good. The normal print pitch is 80 characters per line (12-pitch), but this is a little smaller than you might expect, as the 80 characters are preceded by and followed by a one-inch margin. [Ed. the sample we have prints a typed page exactly the size of those in the Gazette.] It is fast (150 cps) and very quiet, compact (8x11x3 inches). The disposable print-head cartridge pops in or out very neatly. Rather then being an impact matrix printer, the ThinkJet sprays a thin film of ink through a pattern of dots onto the paper. HP says the cartridge is good for 500 pages and although I'm not through my first cartridge yet, that seems to be about it. I have found that the print-head does benefit from an occasional wipe with a tissue, as suggested in the manual. The printer responds fine to WordPro, with the "enhance" command resulting in an underline as I rather hoped it would.

Two small negatives for me: I am accustomed to the knob on Commodore printers that allows you to manually move paper forward or backward to position it. The ThinkJet has buttons for linefeed and formfeed, but of course these only propel the paper forward. Secondly, HP recommends that special paper be used to obtain well-formed and very black printing. This paper costs $90 (Can.) per 2500 sheets fanfold. This is not outrageous but considerably more than the price for ordinary paper. The quality of print is perfectly readable on plain paper, but is definitely darker and sharper on the recommended paper.

All in all, I'm very happy with the performance of the printer and would be glad to pass on information to any interested parties. [Ed. We were sufficiently impressed by this review to try out the printer. It works with WordPro and with PaperClip on available printer files with two exceptions: it won't boldface with any printer file, and it will not proportional-space justify text to the right margin. It would be easy to create a printer file with PaperClip to do boldface, since the printer has a boldface ESCAPE sequence. Right margin justification is not possible for lack of a microspacing command. Yes, underline works fine. The manual is clear and concise. Medium and high-density graphics modes are built-in. It would be extremely difficult to print APL, since a whole row of dots (total width of the paper) must be defined at one time. The price on a 2500-sheet fanfold of special HP paper is $63 U.S. The printer will handle cut paper, also available from HP, though it's slow to load. Available in addition to the default 12-pitch output are compressed print (142 characters per line, expanded (40 cpl), expanded-compressed (71 cpl), all in either normal or bold face, with or without underline.]

COMING UP NEXT ISSUE We promised an article on position independent code (PIC) in this issue, but didn't have space to show how to use PIC in the languages, as we will next issue for sure. John Toebes has written at length on the undocumented routines in SuperPET; we'll begin his material next issue.

Loch Rose and Alain Proulx have written some gorgeous assembly-language programs which load at menu, show a two-column directory, and let you copy, delete, read any SEQ file, and, in general, manage your disk library without ever typing a filename. What was a chore turns into a delight. More coming.

And we just reviewed a new microEDITOR from Joe Bostic, with word-wrap in insert mode (ooooh!), text move (anywhere), echo (duplicate text as often as you wish), ASCII control codes from the mED (you PRINT the files to disk), sorted two column directories, immortal filenames ("names" doesn't change with each "put" or "get" unless you change "name" yourself), a HOME key that homes the cursor, and (at last!), a "dc" command (short for Disk Command) which lets you give any 3.0 DOS command from the mED (goodbye "g ieee8-15")--and an EXEC command which executes, at command cursor, any canned command files you put on disk. SuperED!

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

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

------------------------------Volume II------------------------------

No. 1: $3.75

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

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

APPLICATION FOR MEMBERSHIP, INTERNATIONAL SUPERPET USERS' GROUP

(A non-profit organization of SuperPET Users)

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.
If you send the address label or a copy, you needn't fill in the form above.

For Canada and the U.S.: Enclose Annual Dues of $15:00 (U.S.) by check payable to ISPUG in U.S. Dollars. DUES ELSEWHERE: $25 U.S. Mail to: ISPUG, PO Box 411, Hatteras, N.C. 27943, USA. **SCHOOLS!: send check with Purchase Order.** We do not voucher or send bills.

This journal is published by the **International SuperPET Users Group** (ISPUG), a non-profit association; purpose, interchange of useful data. Offices at PO Box 411, Hatteras, N.C. 27943. Please mail all inquiries, manuscripts, and applications for membership to Dick Barnes, Editor, PO Box 411, Hatteras, N.C. 27943. SuperPET is a trademark of Commodore Business Machines, Inc.; WordPro, that of Professional Software, Inc. Contents of this issue copyrighted by ISPUG, 1984, except as otherwise shown; excerpts may be reprinted for review or information if the source is quoted. TPUG and members of ISPUG may copy any material. Send appropriate postpaid reply envelopes with inquiries and submissions. Canadians: enclose Canadian dimes or quarters for postage. The Gazettte comes with membership in ISPUG.

ASSOCIATE EDITORS

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

Table of Contents, Issue 1, Volume II

ELIZABETH CITY N.C. NOV-9'84
U.S.POSTAGE 54
PB METER 307549

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

First-Class Mail
in U.S. and Canada