A TRIBUTE TO OUR FOUNDING FATHER (?)

Our Secretary, Paul V. Skipski, founded SPUG in the fall of 1982, with a letter to COMPUTE! asking SPET owners to get in touch and get organized. We did. Paul dug up much of our material, got in touch with good people in England, made arrangements for Meizner to get Version 1.1 for you, keeps the books and membership records, builds an RS232 port termination for testing (of which more in later issues), & has some system test programs he's trying to finish for us. All are indebted to Paul. This issue is dedicated to "Pop" Skipski--age 16--a pretty good man.

MULTIPLE SOFT ROMS (IN THE UPPER 64)

(c) 1982, by Roy Busdiecker, Micro Software Systems, PO Box 1442
Woodbridge, VA, 22193

Practically all articles so far in the Gazette have dealt with the 6809 side of the computer. This one deals with how to use the upper 64k of memory from the 8032 (6502/4.0 BASIC) side. (Unless you poke the uppercase Commodore font, all commands in this article are entered in lower case.)

The basic information on how to use 'high memory' is in the manual. That 64k is divided into 16 pages of 4k each, addressed from $9000 to $9fff ($ is a common notation for hexidecimal, or base-16 numbers). You access only one page or bank at a time. The banks are numbered from 0 to 15; you get access by poking the bank number into memory. From BASIC, the command is: POKE 61436, X (where X is the bank number). You can accomplish the same result from the monitor by placing the hex version of the bank number ($00 to $0f) in location $effc (hex version of 61436).

The key piece of information is found in an article entitled Run 64k Programs on the SuperPET, by Paul Donato, in the June, 1982 issue of COMPUTE! magazine. The secret: it is impossible to 'read' the contents of location 61436. If you PEEK (61436) right after you POKE the bank number there, it will look like it didn't 'take'. If you make the change in the monitor, your new entry is followed by a question mark ... the usual indication that you've done something improper. Nonetheless, the banks have been switched. Since the location we change is not really memory, but a register, it acts like a 'write-only memory'...but it does the job.

Want to prove that it works? Here's a way: first, be sure your memory toggle switch (the front one) is set to R/W. Next, run the program to the left.

```
10 for i = 0 to 15
20 poke 61436, i
30 poke 9*4096, i
40 next i
```

It pokes a bank number into the first location of each bank (9*4096 is equivalent to $9000). Now, pick a bank to test, for example, bank 7. In immediate or command mode, POKE 61436,7. When you enter PRINT PEEK (9*4096), the bank number (7 in this case) should appear.

If you own one or more utility ROMs (e.g., SM Kit, Power, Command-O), you can make disk copies from your monitor (more later). Filenames must match those in the program below; it will load each utility in a separate bank automatically. When the program has run, call the utility with SYS 9*4096 (same as SYS 36864).

```
POKE 61436, (bank#)
SYS 9*4096
```

Exit the utility with .X, OFF, or KILL (see program). You switch to any of the others by giving the commands shown at the left. The first line switches banks; the second activates the new utility. If you exit a utility and want it back again (without switching banks), enter SYS 9*4096.

You must have a four-switch SPET to make ROM images (the switches are free at dealers for two-switch models, but labor to install is not). First, unplug SPET; loosen or remove the SuperPET board for access to the sockets. Insert the ROM in socket UD12 (the number is silk-screened on the circuit board). Put the switch for UD12 (3rd from the front) in the ROM position. Fasten the SuperPET board down again before applying power. Leave the board loose AT YOUR OWN RISK.

To copy, apply power (electrical!), put a disk in drive 0, the computer switch in 6502 position, and enter the monitor with SYS 54386. Type in the following command: .S "0:ROM POWER", 08, 9000, A000 (in lower case if you're in the lower case font), and hit <RETURN>. When the image has been saved on disk, exit the monitor with .X <RETURN>. If you have other ROMs to save, repeat the process. Turn power off and remove the last ROM from UD12. Then put the UD12 switch in the RAM position.

I suggest you save the following program as the first on your disk, and then save the ROM images on that disk. If the program below is the first program, you can load and run it (and load all ROM images) with a simple <SHIFT RUN>. (The notice to set the R/W switch will print only the first time this program is run unless you shut down and do a cold start.)

```
200 w=34815:v=peek(w):v=v+1:poke w,v:if v=5 then end
210 if v>5 then poke w,1:v=1
220 if v=1 then print"{clear,down5} set memory switch to {rvs}r/w{rvoff}
    then press {rvs}RETURN{rvoff}"
230 if v=1 then get a$:if a$="" then 230
240 on v goto 260, 300, 340, 380
250 :
260 print "loading sm-kit      (bank 0)     sys 36864 (or 9*4096)   .x
270 poke 61436,0 : load ":rom sm-kit",8
280 rem --- .x to quit ---
290 :
300 print "loading command-o   (bank 1)     sys36864  (or 9*4096)   kill
310 poke 61436,1 : load ":rom-commando",8
320 rem --- kill to quit ---
330 :
340 print "loading power       (bank 2)     sys36864  (or 9*4096)   off
350 poke 61436,2 : load ":rom power",8
360 rem --- off to quit ---
370 :
380 print "set memory switch to {rvs}read{rvoff} then press {rvs}RETURN
    {rvoff}"
390 get a$ : if a$="" then 390
400 print "{down}which bank to you want (0,1,2)? : input s
410 s=int(s):if s<0 or s>2 then 400
420 poke 61436,s
430 print "{rvs}bank"s"{left} selected"
```

MicroCOBOL DOS COMMANDS

Roy Busdiecker also gives us the format for dos commands in COBOL, adding, "Here it is, although I can't for the life of me understand why anyone would want it. This example points out how clumsy COBOL is." Roy feels it more practical to drop into 6502 mode and to issue the commands from BASIC. Another option: save program to disk; then change only three lines near the bottom--two for the character count of

```
*                                   *
* employ dos.                       *
```

```
*
identification division.
program-id. DOS-CONTROL.
environment division.
configuration section.
source-computer. CBM-SUPERPET.
object-computer. CBM-SUPERPET.
input-output section.
file-control.
     select dos assign to 'disk'.
data division.
file section.
fd   dos label records are standard.
01   cmd.
     02 filler pic x(28).     [*]
working-storage section.
01   command   pic x(28).     [*]
procedure division.
     open output dos.
     move 'R0:dos-commands=dos commands' to command. [*]
     write cmd from command.
     close dos.
     stop  run.
```

the command, and the command itself.

The count is found at the notation pic x(n), where n is that count, (of characters in the dos command).

After typing this in (and finding our syntax errors), we know why there's a shortage of COBOL programmers. They're down on the funny farm, or have fled into the night, screaming....

Roy says he's no expert, and that there may be an easier way. Anyone know one?

Asterisks in brackets [*] show the lines which need change, and are not part of program.

ON POKING TO THE SCREEN AND SUPERPET'S GRAPHICS

Terry Peterson, 8628 Edgehill Court, El Cerrito, CA, 94530, one of our members, in an article in December '82 MICRO, points out that the screen POKE codes for the Waterloo font are identical to their PRINT codes (ordinates), and for the printable characters, pure ASCII. Example: poke 32768, 71 will print a capital G at position 1 of the screen (71 is ASCII for capital G). (The method works, if suitably written, for the APL font [p. 85, APL manual]).

In letters, Terry shows that if 32768 is "home" (position 1) on the screen, 32768 + 3*80 must take us to the start of line 4--position 241. So, poke 32768 + 3*80, 73, 84, 32, 77, 79, 86, 69, 83, 33 <RETURN>. Galileo said it. The pokes work both in program & immediate mode. To see the whole Waterloo font on the screen (ordinates 1 through 255), run the program to the left. Since you poke the control ordinates (such as chr$(12)) they won't bugger up the screen by acting as controls.

```
10 ! 'pokescreen'
20 scrn = 32767 ! we add 1 below.
30 print chr$(12)
40 for i = 1 to 255
50    poke scrn + i + j, i
60    j = j + 3
70 next i
80 print rpt$(chr$(10),17)
```

Note that the reverse field graphics we get on the shifted keypad have their non-reverse counterparts from chr$(1) through (11). Chr$(0) and many ordinates from chr$(12) on print a small square. Terry advises not using ordinates above (11) for poked graphics (the keypad graphics are another matter) because of problems in programming to save the graphics.

This is as good a time as any to demonstrate the speed advantages of integer arithmetic on SPET. Below find two programs, identical except that one uses integer arithmetic to the hilt, and the other does not. The run time is cut almost in half with integers. If you revise the programs, you'll find out which of the integer arrangements is most effective and which is marginal (examples:

changing the j loop to j% saves only one second; changing to i% in lines 50 through 70 has major impact, as does converting 32767 to scrn%).

```
10 ! 'poke reverse'
20 a$=" Try reverse printing "
30 v=time
40 for j = 1 to 100
50   for i = 1 to len(a$)
60     poke 32767+i,128+ord(a$(i:i))
70   next i
80 next j
90 print "Elapsed time: ";time-v
```

```
10 ! 'poke reverse%%'
20 a$=" Try reverse printing "
30 v=time : x%=len(a$) : scrn%=32767 : y%=128
40 for j% = 1 to 100
50   for i% = 1 to x%
60     poke scrn%+i%,y%+ord(a$(i%:i%))
70   next i%
80 next j%
90 print "Elapsed time: ";time-v
```

(32767 is supposed to be maximum for integers, but these programs run without error)

Time to run: 56 seconds

Time to run: 29 seconds

A SCREEN DUMP FOR POKED GRAPHICS TO MX80 PRINTER

by Terry Peterson

For the poked graphics (ordinates 1 through 11) there's a way to save to printer, written by Terry. It comes in two parts: proc gdump, which is an immediate mode procedure, & fun_gform$.

```
62150 ! MX80 Graphics Screen Dump for SuperPET (with GRAPHTRAX+ )
62160 proc gdump
62170 open#12, "ieee4", output                    ! Printer file.
62180 for ii = 32767 to 32767+23*80 step 80       ! Do 24 lines
62190   for mm = 1 to 80                          ! of 80 characters
62200     aa$(mm:mm) = chr$(peek(ii+mm))
62210   next mm
62220   if str$(aa$,1,4) = "quit" then quit
62230   print#12, fn_gform$(aa$)
62240 next ii
62250 print#12 : close#12
62260 endproc
62270 !
62280 ! Change SPET graphics to MX80 equiv.
62290 def fn_gform$ (xx$)
62300   restore
62310   yy$ = xx$
62320   for jj = 1 to 11
62330     read MX_char
62340     while idx(yy$, chr$(jj))
62350       kk = idx(yy$, chr$(jj))
62360       yy$(kk:kk) = chr$(MX_char)
62370     endloop
62380   next jj
62390   fn_gform$ = yy$
62400   data 156,157,154,153,134,149,158,150,152,151,159
62410 fnend
```

Terry comments: "This version works with characters 1-11 & could be used by modifying fn_gform$ appropriately to handle reverse video on printers that will print in reverse field which the MX80 will do only in dot-graphics."

He adds: "This solution isn't very good because there may be other data statements. Nevertheless, it serves to illustrate the method."

If "idx" throws you, see p. 202 of the M-BASIC manual--an elegant way to find 1 to 11 & to substitute MX80 code.

DRAWING AND SAVING KEYPAD GRAPHICS TO DISK

Terry Peterson also gives us the key on how to save keypad graphics to disk (plus alphanumerics in normal or reverse field), and how to get the image back, either in immediate mode or from program. Find below three immediate mode procedures to draw, save, and retrieve images.

Call **proc draw** to draw from the shifted keypad and to print alphanumerics with them. For reverse field alphanumerics, touch <REVERSE>. To leave reverse field, touch <REVERSE> again. Exit 'draw' with <UP-ARROW>. SAVE the screen image with **scrn_save** (the large loop takes a while, so be patient). When you want the image back, call **scrn_get**, which is fairly fast. You can save graphics, whether POKEd or PRINTed, with **scrn_save**, originally written by Terry for the lower, poked graphics, as was the original **scrn_get**, which we've changed for the keypad. Terry's **scrn_get** for the low, poked graphics didn't work because of a bug in Verson 1.1 (below). We'll cover a scrn_get for poked graphics next issue.

```
60000 proc draw ! packet is 'pixtrial'
60010 if cursor(1841) then print "Clear screen, draw. Up-arrow to exit"
60020 loop
60030   get i : if i = 0 then 60030
60040   if i = 94 then quit          ! Up-Arrow exits procedure.
60050   if i = 255                   ! Hit reverse key to print text in
60060     rvs = rvs + 1              ! reverse field. Hit again to get out.
60070     if rvs > 1 then rvs = 0    ! Program reverses only alphanumerics.
60080   endif
60090   if rvs and i >30 and i <127 then i = i + 128
60100   if i <> 255 then print chr$(i);    ! Watch that semicolon
60110 endloop
60120 endproc
60130 !
60140 proc scrn_save                 ! Material should start on first line
60150 aa$ = ""                       ! of the screen or near it to save time.
60160 if cursor(1761)                ! Proc saves all graphics.
60170    print chr$(6);
60180    input "How many lines to be saved? ", nn%
60190 endif
60200 scrn = 32767 : x% = 80
60210 for ii% = 1 to x%*nn%-1        ! 32767 is maximum for integer arithmetic.
60220   jj% = peek(ii% + scrn)       ! Overflow errors if scrn% is used.
60230   aa$ = aa$ + chr$(jj%)        ! There's a way around this with negative pokes.
60240 next ii%                       ! Article on it, next issue.
60250 open #14, "picture", output
60260 print #14, aa$ : close #14
60270 endproc
60280 !
60290 proc scrn_get                  ! For keypad graphics and text only
60300 on eof ignore
60310 print chr$(12) : open #11, "picture", input
60320 loop
60330   linput #11, aa$
60340   if io_status <> 0 then quit
60350   print aa$;
60360 endloop
60370 close #11 : if cursor(1761) then print "Finished"
60380 endproc
```

A PATCH FOR MICROBASIC VERSION 1.1

Waterloo (WCS, Ltd.) has issued a patch because a bug in the 'open' code causes a recordsize of 80 to be assumed. Try to print an 80-character string, and the 80th is replaced by a carriage return; the 80th character

```
10 on eof ignore
20 open#2, "disk/1.BASIC,PRG",input
```

(cont. from prev. page)

```
 30 open #3, "disk/0.BASIC,PRG", output
 40 x = peek(86)*256 + peek(87) + 4
 50 y = peek(x)*256 + peek(x + 1) + 1
 60 poke y, 0, 0
 70 offset = 26*512 + 182
 80 for i = 1 to offset
 90   get #2, a$
100   if a$ = "" then a$ = chr$(0)
110   print #3, a$;
120 next i
130 get #2, a$
140 if ord(a$) <> 80
150   print "wrong bye to patch"
160 else
170   print #3, chr$(0);
175   loop
180     get #2, a$
190     if (io_status = 2) then quit
200     if a$ = "" then a$ = chr$(0)
210     print #3, a$;
220   endloop
230 endif
240 close #2 : close #3
```

prints on the next line. Put a backup language disk in drive 1; a disk to copy to in drive 0. Then run the patch. Take the dog for a walk. Program runs 40 minutes. Patched version's on disk 0.

While the patch cures the problem above, it seems to have a bug also, sometimes generating a carriage return to printer in the midst of a print statement using semicolons: (print "Patch";n;"now") puts n on the next line. This bug does not show in the same print statement using V1.0 or unpatched V1.1. We've written Waterloo for help. Stick with 1.0?

ADVICE NEEDED! ADVICE NEEDED!

Don Momberg, 206 Green Brook Rd, Green Brook NJ 08812 (201-752-8604) writes that when his SuperPET died, he got the 8032 board replaced (and it works fine), but 6809 mode gives 25 white horizonal lines and a blank cursor going through the motions of printing the menu, but no characters appear. If he pokes the Waterloo font from the lower 32, he gets the white lines again. If he pokes to Commodore fonts, all's okay. His dealer can't help. Send a cure or suggestions to Don DIRECTLY, copy to us.

BITS, BYTES & BUGS

by: Gary L. Ratliff, 1033 Hattiesburg Drive, Magee, MS 39111

Welcome to the first of a series of articles on assembly language programming with the 6502 and 6809 processors found in SuperPET. I haven't yet obtained the promised Development system for the 6502 from Waterloo Computing Systems, but when it is released I'll cover it as well as the Assembly Language Development package for the Motorola 6809 chip.

Each article will be illustrated with examples. At least one known error will be presented in each. Programming is learned not by reading but by programming; if you enter the examples you should discover the error; when you do, the correct principle will become part of your programming skills.

The editor asked me to start with fundamentals since members come from a highly diverse background. This series begins with the a b c's of assembly language programming and will continue through more complex techniques. If you're serious about learning assembly language, you'll want to add some books to your library: 6809 Assembly Language Programming and 6502 Assembly Language Programming, both by Lance A. Leventhal, published by Osborne/McGraw-Hill. I suggest you subscribe to COMPUTE! magazine (PO Box 5406, Greensboro, N.C. 27403). [Ed: Add MICRO (MICRO INK, 34 Chelmsford St., PO Box 6502, Chelmsford, MA 01824) if you can. SPET gets more support from MICRO than any other US magazine.]

The DEVELOPMENT system loads when you hit d <RETURN> while the main menu is on screen. If the system fails to load, check the UD12 switch. It must be in RAM

position. DEVELOPMENT is composed of a series of programs which work together powerfully to create assembly language programs:

The **ASSEMBLER** takes instructions you write, such as LDD STA BRA SEX, and directives such as EQU END FCC RMB, and forms a series of binary instructions which the processor can execute. The assembler solves the **which-code** problem; it writes two programs: (1) a list of your source code (assembly language) and the resulting hex code for people to read, and (2), an object code (machine language) program, for the linker to process and locate.

The **LINKER** is mainly concerned with **where** to locate the program instructions. It also writes two programs: (1) a map of where the program is located, and, (2) a module (mod) which you load and run. It writes a third program: an 'exp' file, if you use any built-in routines or 'exports' from other programs.

Your instructions to Assembler and Linker are written (and rewritten) in:

The **EDITOR**, a tabula rasa on which you, the programmer, set forth **what** steps should be taken to solve a problem.

The **MONITOR** deals with **when:** no problem is solved until your program is executed (and probably debugged) in the MONITOR.

Finally, the **why** aspects are solved by the programmer, who must view the processor and other hardware as a black box which it is his task to define.

We now concentrate on the **EDITOR** and how to use it easily. (In the examples below, my comments are enclosed by brackets: [ ]. Do NOT include them in the file. You can add comments to an **asm** file, but only be prefacing them with a semicolon.)

To save a program named 'file' for the use of the assembler and the linker, the commands are:

```
p file.asm [A file with the .asm suffix is automatically sought by the
            assembler.]
p file.cmd [A file with the .cmd suffix is used by the linker.  The 'p'
            command 'puts' the file to drive 0.
```

If the assembler or the linker present any error messages, the proper file must be recovered and corrected. Get the files by substituting a 'g' (no quotes) in the commands above. (No quotes at all are employed in the commands.)

We now show how the program: "EXAMPLE 1" from the DEVELOPMENT manual may be entered and executed without using any of the EDITOR's PF (programmed function) keys. Load DEVELOPMENT; select the EDIT function, and type in:

```
i           [The 'i' places the editor in input mode, and automatically
             opens up lines in the editor. If you use PF0 (SHIFT 0) to
             open lines, you must hit it every time you want a new line.
             Be sure to hit <RETURN> after each entry, below.]

;example 1 [The semicolon prefaces a comment which is ignored by the
            assembler program.]

lda #'a     [ld mean load; #, immediately; a is the 'a' accumulator; we
             put 'a' there in this, the first line of our 'source' code]

sta $8000   [store contents of 'a' accumulator in $8000 (decimal 32768),
             which is the first position (home) of the screen.]
```

```
swi          [The swi means 'software interrupt'; you pass control from
              the program back to the monitor. Last of source code.]

end          [The end must be included as the last instruction.]

.            [The period, entered in the input mode, causes a return from
              screen mode to command mode. It must be the only character on
              the line. (The period will be deleted from the file automat-
              ically.) For this feature to work, you must not use the up or
              down cursor control keys, which will cancel input mode.]

p ex1.asm    [The program is complete and is put on disk as a file named:
              'ex1.asm', which will be used by the assembler program.]

*d           [This deletes all text from the editor; you now have a clean
              slate to create the cmd file for the linker.]

i            [Insert mode again.]

"ex1"        [First line of the cmd file for the linker; it must be
              named, and the name must be enclosed in quotes.]

org $1000    [Tells the linker where to place a program in memory.
              'org' means origin, or where the program starts.]

"ex1.b09"    [Tells the linker what to place there: the object (machine
              language) code which will be created by the assembler when
              we actually assemble code, below. Files created by assembly
              and linking will be examined next issue.]

p ex1.cmd    [The cmd file is complete, so put it on the disk.]

<RETURN>     [Exit the editor, return to menu. We now 'run' our programs.]

a            [This calls the assembler program from the development menu.]

ex1          [This is the name of the program we want assembled.  Don't add
              '.asm' to the program name. Hit <RETURN> and assembly begins.]

<RETURN>     [Exit the Assembler and return to menu.]

l            [Lower case 'L'. This calls the linker program from the menu.]

ex1          [Again we use the name of the program we want linked; again we
              don't add '.cmd' to the program name.]

<RETURN>     [Exit the Linker and return to menu.]

m            [This calls the monitor program from the menu.]

l ex1.mod    [Lower case 'L' loads the object program created by the linker.]

g 1000       [The g means go (run). We stored the code at $1000 with the org
              command. This executes code starting at $1000. Look closely.
              Find an 'a' at home position, where the W of Waterloo was?]
```

```
q          [We exit the monitor with q for 'quit']
```

In this installment we abandon an alphabet of languages at our beck and call to print an "a" in the upper left corner of the screen. But, instead of being limited to the commands of a formal language, we are able to communicate with the processor in machine language. And, as we will later see, that language is powerful and fast. If you have trouble or comments on this column, please feel free to write to me at 1033 Hattiesburg Drive, Magee, MS 39111. Bye for now.

A SHORT, STRUCTURED SCREEN DUMP
by Terry Peterson

As promised last month, we print Terry's screen dump, a simpler, neatly structured, and straightforward version of the dump printed in Vol. 1, No. 2.

```
62000 proc dump
62010 open#12,'ieee4',output
62020 open#13,'terminal',input
62030 open#14,'keyboard',output
62040 print chr$(1)
62050 for ii = 1 to 24
62060   print#14,chr$(13)
62070   linput#13, aa$
62080   if aa$ = "quit" then quit
62090   print#12, aa$
62100 next ii
62110 print#12:close#12:close#13:
      close#14
62120 endproc
```

* * * *

ON ABSOLUTE CURSOR CONTROL

A number of readers noted that we had not included the first method below in the discussion of absolute control.

There's a second, absurd way which works. The first is handy when you want to move the cursor without a print statement. Both methods print "Sample", starting at screen position (n):

```
No. 1: x=cursor(n) : print "Sample"
No. 2: poke cursor(n),1 : print "Sample"
       (1 is a dummy argument)
```

RELATIVE CURSOR CONTROL

We had meant to print Dan Horn's article this issue, and reserved a page for it and an article by John Spencer on how to set tabs from program. But John's article, upon arrival, was longer (and held much more information) than expected. Since we have insufficient space, we postpone both to the next issue. Unexpected pearls!

WRITERS! We need more contributions which define the machine. Be brief; write in present tense; avoid passive voice. We have nine pages (no more), so distill to the essence of the essence. Be liberal with examples. If an article must be long, break it into parts for consecutive issues. Keep articles (or parts) to two pages or less (54 lines per page, 80 characters per line, including white space). If you're no writer but have something important to say, send the material anyway. An editor's job is editing.

****

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

APPLICATION FOR MEMBERSHIP, SUPERPET USER'S GROUP

Name:_____ Disk Drive: _____ Printer:_____

Address:_____

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

Enclose Annual Dues of $10:00 (U.S.) by check or money order, made out to Secretary, SPUG. Overseas dues: $20.00 U.S. Mail to: Paul V. Skipski, Secretary, SuperPET User's Group, 4782 Boston Post Road, Pelham, N.Y. 10803, USA.

Newsletter published by the SuperPET Users' Group (SPUG): editorial offices at PO Box 411, Hatteras, N.C. 27943. Secretary, Paul V. Skipski, 4782 Boston Post Road, Pelham, N.Y. 10803. Membership applications and inquiries to Mr. Skipski. Newsletter material to Hatteras, attn: Dick Barnes, Editor. Enclose a self-addressed, postpaid envelope with all material submitted and all inquiries requiring reply. Membership: $10.00 per year, **U.S.** in North America, $20.00 overseas and elsewhere. See enclosed application.

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 Canada, U.S., most European nations, Russia, China, and most of Araby. Each Gazette issue weighs one ounce: 20 cents U.S. & Canada, 80 cents to Europe. For other rates, see your local post office.

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