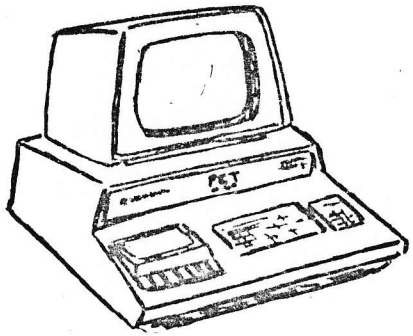
BOX 43
AUDUBON
PA 19407

THE PET PAPER

BOX 43
AUDUBON
PA 19407



VOLUME 1 ISSUE 1

EDITORIAL

One of the most exciting events in life is starting off on a new adventure, and we are both happy to be starting on this one -- the PET PAPER.

We know that many of you are also starting a new adventure, just entering the fascinating world of personal computing. We're looking forward to being part of your new world -- pointing out interesting sights and warning you of hidden pitfalls.

We also realize that many of you are experienced travelers in the land of The Computer, and for you we offer the camaraderie of the open road, a chance to swap tales and information, to compare notes with others who have traveled the same path.

Many of our concepts for the PAPER came from another publication, the KIM-1 User Notes, edited by Eric Rehnke. As you look through this issue, you'll see that we've tried to provide much more than mere technical information, as important as that is. Of special interest to many of you will be the software we offer. Here's a chance to expand your own library (at almost no cost if you have programs you're willing to share), to find people who are at your level of interest and expertise, to find out who is planning to build the item you've been looking for, and to get around the scarcity of available information.

A lot of what we hope to accomplish depends on you. What do you want to know? What have you been able to find out? What are you doing with your new PET? We want to hear about your expertise, your programs, your local User Group. We would be very surprised if something you read in this first issue doesn't tempt you to write us with a question, an answer, a comment, or information we should have. So DO it!

The PET is an incredibly powerful tool, and we're convinced that the PET and its competitors will have a very real, very positive, and very large impact on our society. But for that to happen quickly, the owners and users of PET (as well as other consumer-oriented computers) need to be able to give and receive information, to share experiences, thoughts, software, and expansion plans.

The PET PAPER is dedicated to that task.

Rick Simpson

Terry Laudereau

Improvements

Figure 4 shows some "extras" you can add to the circuit. SW1 removes power from the circuit. When PET is first turned on, the user port pins have enough voltage to turn on the oscillator, even though they are programmed as inputs, so you may wish to keep the oscillator turned off until your program has initialized the user port. R1 replaces the 68K resistor with a 50K potentiometer so you can vary the frequency of the oscillator. R2 is a 10K potentiometer to vary the volume of the speaker.

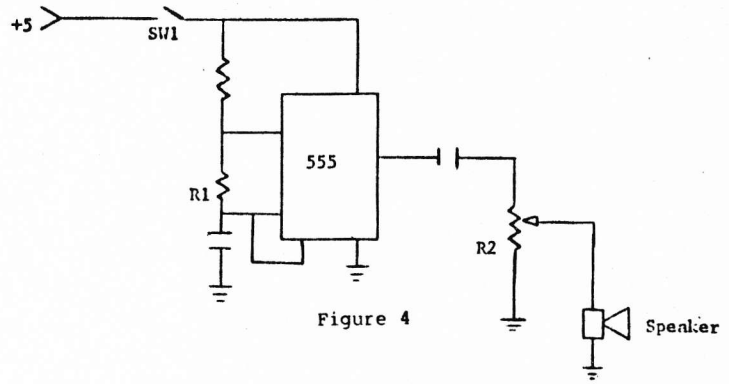


Figure 4

SUBSCRIPTIONS, ADVERTISING,
and Other Money Matters

The PET PAPER is published ten times per year. Each ten issues constitutes a volume, and subscriptions are for one volume only. Your subscription will not span two volumes, so if you subscribe at issue #8, volume 1, you'll still receive all ten issues of that volume.

You may, however, purchase a copy of each issue separately for $2 each.

If, during any publication year, you wish to buy back issues, send us $15 per volume (or $2 per issue) and we'll ship your order ASAP.

Subscription rate increases will not occur during any given volume, and subscribers will be notified if the rates are being increased for the next volume.

Advertising rates are:*

1/4 Page --	$ 50	
1/2 Page --	90	*For other rates,
1 Page --	150	please write.

Payable with the camera-ready copy for the ad. Ads not accompanied by payment will not be published. If we misprint an ad, the next issue will carry the same ad at no cost, provided the advertiser notifies us within 15 days of the mailing date of the issue in which the error occurred.

port for other purposes, slightly different programming will be required.

Finally, to turn the oscillator on when we want to bark, use the statement:

```
POKE 59457,1
```

which sets bit zero of the user port high and, of course:

```
POKE 59457,0
```

when we want to turn the oscillator off again. So a short program to make a "bit" would look like:

```
10 POKE 59459,255:REM SET AS OUTPUT
20 POKE 59457,0:REM SET OUTPUT LOW
30 POKE 59457,1:REM SET OUTPUT HIGH
40 A=SIN(2):REM WASTE TIME
50 POKE 59457,0:REM SET OUTPUT LOW AGAIN
```

What is line 40 for? Just to waste time so PET doesn't turn the oscillator on and off so fast you hardly hear the "bip."

Construction

I built my test circuit on an AP Products breadboard. You will probably want to use a small piece of perf board glued right to the connector you use to connect to the user port connector on your PET. You may have trouble finding the 12-position user port connector and the 6-position connector used for the second cassette. (We need this to get +5V, not available on the user port connector.) What I did was to get a 44-pin (22 position) edge connector (Radio Shack again) and carefully cut it with a fine-tooth saw to get the two connectors I needed. The connections look like this:

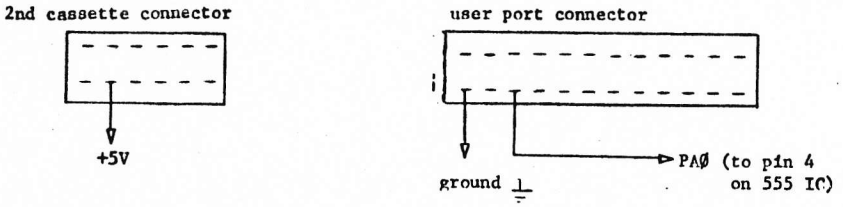


Figure 3
Back View of PET Connectors

PET PAPER FLEA MARKET--SOFTWARE EXCHANGE LIBRARY

Have you written a neat but trivial program for your PET? Or transcribed a public-domain program from some other BASIC to PET BASIC? Send us the tape! We'll send you a coupon good for any other FLEA MARKET program in our library if we accept the program. When we have a program in the library that you want to use, send us back the coupon (and $2 to cover duplication costs and postage) and we'll send you the program you want.

If you don't have a program of your own, but you see something in our library that you want, send us $5 -- we'll send you the program you specify!

Here's a list of the current FLEA MARKET programs:

Flea Market Software Exchange

1. Addition Game--A fun game with 10 levels of addition skill.
2. Othello for One--The famous Othello game, but this one plays a good corner game as well!
3. Othello for Two--PET does the work and keeps score.
4. Codes--PET selects the Codes, you're decoder.
5. Road Race--Race any of 5 vehicles, don't run out of gas and don't get hit by a bus!
6. Slot Machine--As fun and as tense as the real Las Vegas style slots -- but you keep your money in your pocket.
7. Useful Routines--GET (with a cursor) instead of INPUT. X$, Y$ for cursor control -- Ball demo -- Frame your PET -- Simple sort -- Forms and others.
8. Address File--Put your mailing list on tape!
9. States and Cities--Do you know all 50 states and their capitals? This game teaches and drills.

PET PAPER SOFTWARE SHELF--SOFTWARE SALES

Have you written something significant or original that you'd like to share with others but which requires more than average expertise to write? Send us your tape! If we agree, we'll send you a contract specifying royalty terms (normally 20% per program). All royalties are paid quarterly and are based on the retail price of the tape.

Do you see a program on our Software Shelf that you'd like to use? Send us the purchase price and we'll ship you the tape. All programs on our Software Shelf are adequately documented.

Here's a list of Software Shelf offerings:

Software Shelf

Deflection $10.00

One of the best PET games we've seen, full of fast-paced action involving targets, deflection shields, and you!

Data Retrieval for the Home $10.00

Keep all your valuable items listed on tape. A true file-keeping system.

Stat $20.00

A real statistical package for up to 100 data points -- basic stats -- linear regression -- moving averages -- plotting -- transformations.

Software Guarantee:

Because the PET's recording system has been altered somewhat since the PET was first introduced, a program tape may not load properly on a very small percentage of PETs. If the tape you receive from us does NOT load, send it back! We'll re-tape the program on the old recording system and ship it to you within a week.

New Product Announcement

RS-232 PRINTER ADAPTER FOR THE COMMODORE PET

Connecticut microComputer announced the first in a line of peripheral adapters for the COMMODORE PET. The PET ADApter model 1200 drives an RS-232 printer from the PET IEEE-488 bus. The PET ADA 1200 allows the PET owner to obtain hard copy program listings, and to type letters, manuscripts, mailing labels, tables of data, etc., using a standard RS-232 printer.

The PET ADA model 1200 is available assembled and tested, without power supplies, case, or RS-232 connector for $98.50 (plus $5.00 for shipping and handling) or complete for $169 (plus $5.00 for shipping and handling). Specify baud rate when ordering. (300 baud is supplied unless otherwise requested.)

can only turn an output port on and off about 100 times a second. This means that the circuit of Figure 1 would only produce at best a tone of 100 Hz -- just a loud hum.

To circumvent this problem, we won't make PET make the noise; instead, we will have PET just control a "noisemaker" -- a little circuit which will produce a tone as long as PET tells it to.

Figure 2 shows the schematic diagram for PET's "voice."

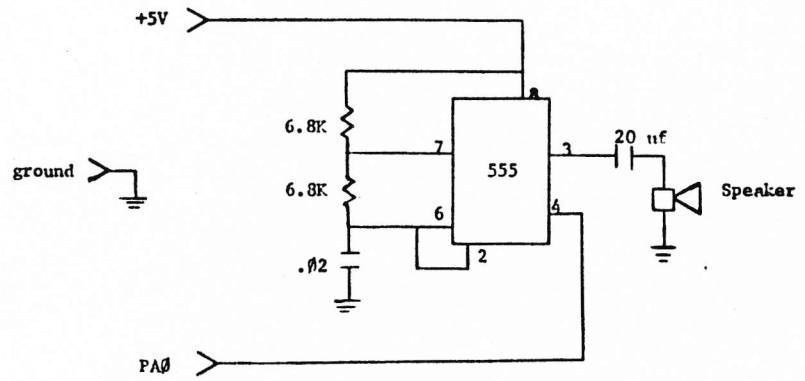


Figure 2
PET-Controlled Noisemaker

The circuit is built around a type 555 integrated circuit timer wired up as an oscillator to drive the speaker. Whenever pin 4 has more than about 2 volts on it the circuit will produce a loud tone from the speaker. Lower the voltage on pin 4 and the tone will be turned off -- it's as simple as that!

Software

In order to allow PET to control the oscillator, only a few simple program steps are required. First, since the PET user port is set up when you turn it on to read from external crcuits, not control them, we have to change it froman input port to an output port. This is done by a program statement:

```
POKE 59459,255
```

which changes the value in the user port "Direction Register." This has to be done only once in your program.

Next, we want to make our newly created output lines (PET has 8 of them) all go to their low state so the oscillator is turned off. We do this through the statement:

```
POKE 59457,0
```

which puts zeros in all the bits of the user port "Data Register." Those of you who are knowledgeable about PET hardware will realize that the above two statements are programming "overkill" since we have initialized and cleared ALL the user lines, not just the one we need to control the oscillator. If you are using other lines on the user

TEACHING YOUR PET TO BARK

There is no doubt that PET has many applications in business and industry, but for many of us the best part of PET is its ability to play games. PET'S graphic capability, ease of use, and high-speed operation make it ideal for endless "video games," but every PET delivered so far has one glaring problem when it comes to programming games -- PET is mute! Except for the 15,000 cycle whine of its horizontal oscillator in the video display and the faint click of the keys as you respond to its question, your PET is about as noisy as a pet rock. Think of a PET programmed as a pinball machine -- how dull to watch the "ball" bounce off the bumpers without all the fun noises of the real thing!

Teaching your PET to make noise is really very simple -- all you need are 2 resistors, 2 capacitors, a small speaker and a connector -- all available at your nearby Radio Shack store! (While you're there, take a look at the TRS-80 and find out why you're lucky you bought a PET!)

If you are getting impatient, look at Figure 2 which is th schematic for PET's electronic voice.

In many microprocessors such as the KIM, you can make "beeps" and "boops" by rapidly turning an output line on and of and connecting the line through an integrated circuit buffer to a speaker. (The buffer is necessary because most output lines can't provide enough power to drive a speaker.)

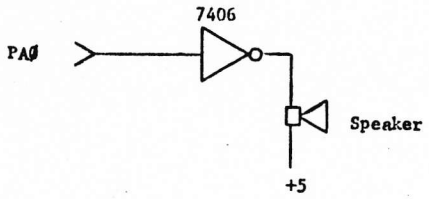


Figure 1
Microprocessor Speaker Output

PET has such output lines (the user port) and, if you are willing to do machine language programming, Figure 1 shows all the hardware you need to make music on PET. However, since most of us bought a PET to avoid machine language programming and would rather program in BASIC, we are going to have to add more extra circuitry. The reason for this is the fact that, even thoughPET's BASIC interpreter is very fast, it

SOFTWARE SHELF: Please send the following program tapes. Payment in full enclosed.

_____ _____

NAME_____ Cash, check, or money order $_____

ADDRESS_____ MC VISA (Circle one) #_____

CITY, STATE, ZIP_____

Mail to: THE PET PAPER Master Charge Interbank #_____

P.O. BOX 43 MC/VISA Expiration Date_____

AUDUBON, PA 19407

FLEA MARKET: Please send the following program tapes. Payment in full enclosed.
*I enclose_____ programs instead of payment_____

_____ _____

NAME_____ Cash, check, or money order $_____

ADDRESS_____ MC VISA*(Circle one)#_____

CITY, STATE, ZIP_____

Mail to: THE PET PAPER Master Charge Interbank#_____

P.O. BOX 43 MC/VISA Expiration date_____

AUDUBON, PA 19407 *CREDIT CARDS ACCEPTED ON ORDERS OVER $10.00

PET PROSE: Please publish my name as an author of specific application software. I enclose $25.00 for publication in 10 issues (one volume) of the PET PAPER.

NAME_____ APPLICATION FIELDS_____

ADDRESS_____ _____

CITY, STATE, ZIP_____ _____

_____ cash check money order MC VISA (circle one)

Mail to: THE PET PAPER MC/VISA #_____

P.O. BOX 43 Interbank #_____ Exp Date_____

AUDUBON, PA 19407

SEQUENTIAL PROGRAM STORAGE

Your PET can do many wondrous and magnificent things with cassette tapes, but finding a blank space in which to save your next program is not one of them.

When you type "SAVE PGM 4" and press RETURN, PET starts right off, saving PGM 4 with all its might. Unfortunately, if you didn't have the tape properly positioned, PET destroyed PGM 1. It saved PGM 4 right over it. Even having destroyed a week's work, PET is utterly incapable of understanding why you sit there beating your forehead against its CRT housing.

So, let's save PET from bewilderment and you from frustration.

Once PGM 4 is in memory, rewind your cassette tape and type VERIFY PGM 3. (You see how sneaky this is? PET will find PGM 3 and try to verify it against PGM 4 in memory.)

Eventually PET will announce sorrowfully, ?VERIFY ERROR. And it's correct, of course. But now the tape is properly positioned at the beginning of a blank space on which you can save PGM 4!

Press the STOP button on the cassette unit, then type SAVE "PGM 4" and hit RETURN. Now PET will happily take charge and issue appropriate instructions for you. When PET says it's ready, rewind and VERIFY PGM 4.

Good luck!

PET 0010430 CALLING...

There's a lot of speculation and dreaming (and maybe just a little planning and designing) for a vast, long-distance communications network for personal computers. That includes us, folks!

Seems like the biggest problem is the standardization of signals between one brand of computer and another. But then there's the problem of data storage (What happens if all the lines are busy? Or if your computer isn't taking messages today?) and data transfer (Someone has to route all this traffic!).

Some of the popular plans are based on the telephone network (Everyone has a telephone -- but the bills could be astronomical!) or on amateur radio, which means each participant would have to learn Morse Code and get an FCC license to let his computer out into the air.

But think of the possibilities! Your PET could be programmed to plan your menu, keeping track of every item used, check with you to be sure you did follow the menu, re-order for you when your supplies run low, transfer funds from your bank account to the supermarket, and print on its daily calendar, "STOP AT MARKET -- PICK UP ORDER #102348." (If someone will hurry up and build a robot...)

And imagine half the personal computers in the country playing simulated war games -- new strategies and defenses. (On second thought, we'd all get classified as TOP SECRET.)

A long-distance communications network is a highly desirable step, no doubt. When it will happen is anyone's guess, but we'll keep you posted as we learn more about it. Meantime, you'll find articles in the February 1978 issue of BYTE which discuss this concept in great detail.

```
100 FOR I=1 TO 10:A(I)=I:B(I)=I+1:NEXT I
110 A(5)=B(5)
```

* * * * * * * *

Pet does not require closing quotes in PRINT statements when the print message is at the end of a line.

```
10 PRINT"HELLO, FRIEND
```

requires no closing quotes, but

```
10 PRINT"HELLO, FRIEND";
```

does -- because of the semicolon.

* * * * * * * *

It is not necessary to separate items in a PRINT list with semicolons, either.

```
100 PRINT"SUE JONES"A*3"TOMORROW"L$
```

and

```
110 PRINTTAB(10)"HELLO"1234TAB(30)"*"5678
```

works too. You get semicolon spacing (i.e., a cursor-right) between numerics and no blanks between strings.

* * * * * * * *

PET prints numbers in this format:

SIGN-SPACE NUMBER CURSOR-RIGHT

which is nice -- because you can print numbers right up to the edge of a pre-drawn form on the screen.

* * * * * * * *

To get a list of all the programs saved on one tape, type 10 PRINT then type VERIFY"?". PET will search the tape for a program named "?" and will list everything it finds while searching. Allow no more than 7 minutes to elapse after each program name is displayed. If PET hasn't found a program within 7 minutes, there are probably no more programs on the tape.

* * * * * * * *

Press the reverse (RVS) key to slow down the scrolling during a LIST.

STANDARD SYMBOLS FOR PET FUNCTIONS

People around the country are crying for a standardization of symbols to depict the PET special function keys, but no one has done anything about it. Here are the symbols we plan to use in THE PET PAPER. We'll reprint them in every issue so you'll always have them handy.

Ⓗ	Cursor HOME
Ⓒ	CLEAR Screen
Ⓡ	Cursor RIGHT
Ⓛ	Cursor LEFT
Ⓤ	Cursor UP
Ⓓ	Cursor DOWN
Ⓕ	Reverse FIELD on
Ⓞ	Reverse field OFF
ⓐ	Carriage Return
↑	Shift On
↓	Shift Off
/	Delete
Ⓘ	INSERT
◯	Space
*	Stop

Here's a sample program to show you how the code looks:

```
10 PRINT"Ⓗ";          Send the cursor HOME
20 FOR I=1 TO 10
30 PRINT"Ⓓ"           Print "cursor down" 10 times
40 NEXT I
50 FOR I=1 TO 10
60 PRINT"Ⓡ";          Print "cursor right" 10 times
70 NEXT I
80 PRINT"HI!"         Print a message
```

We chose these symbols because they are easily drawn, typesettable, and some of them are mnemonic. If you have a better idea, let us know about it!

INTRO TO BASIC

BASIC is a language intended for use by beginning computerists. How it was invented, where, when, why, and by whom, is a topic for someone else to cover -- we are going to just USE BASIC to communicate with the PET.

BASIC is very much like English. It's less confusing than English, more organized, and its rules are more rigid than English syntax rules. Let's look at the word RUN. In English, the word RUN can be used many different ways. Look at all the things RUN means in these sentences:

1) I have to run (turn on) the dishwasher tonight.
2) George runs (operates) a small grocery store in Des Moines.
3) Sam, will you run (drive) down to the post office for me?
4) Ellie wants to run (contend) in today's track meet.
5) Tom just hit a home run (unit of scoring).

In BASIC, the word RUN means "to execute" a program. That's ALL it means!

LIST, in English, can mean (a) an arena for a contest, (b) a roll, or catalog, (c) to tilt or a tilt, (d) to hearken, or listen. In BASIC, LIST always and only means, "display the lines of code for this program."

So we can see that BASIC will be easier to learn than English, as far as the word definitions go.

Then there are punctuation and spelling rules. In English, a sentence starts with a capital letter, contains a subject, a verb, and an object, and ends with a period. In BASIC a statement line starts with a line number, contains a keyword* and one or more operands,* and ends with a carriage return. In English, if more than one sentence occupies the same line, the sentences are separated by two spaces; in BASIC they are separated by a colon.

One more very important comment before we start. In English, if you misspell a word, or use incorrect grammar, the chances are that you'll still be understood. People can pick up your intention from the context of your whole paragraph of written or spoken words because people are much, much smarter than a computer. If you misspell or misuse a word in BASIC, PET will NOT understand you. What's more, it prints a message on the screen (?SYNTAX ERROR) telling you you blew it -- and then lets you guess how and where you made your mistake!

Okay -- now let's get on with learning BASIC. In this article, we're going to talk about LINE NUMBERS and 8 keywords: READ, LIST, DATA, RESTORE, PRINT, IF, GOTO and RUN.

---

*At the end of this article is a BASIC dictionary defining the *'d words.

TRICKS TO HELP TRAIN YOUR PET

Because people like to build grandiose coding structures, we often try to conserve memory space by concatenating statements and eliminating as many line numbers as possible. And programs which worked quite well before statement concatenation suddenly bomb. Why? The code's the same -- only the line numbers have been changed to protect memory space. So what happened?

One possibility is that statements were concatenated following an IF/THEN statement, like this:

```
100 IF A=B THEN PRINT"A=B":GOTO 2000
110 PRINT"NO MATCH":GOTO 1000
```

In line 100, if A is equal to B, the program will print "A=B" and then go to 2000. If A is not equal to B, PET stops scanning the line before it reaches the keyword "THEN" -- so it won't print "A=B" and it won't go to 2000. Instead, it will execute line 110. Sounds fairly straightforward, doesn't it? But, in an effort to save space, suppose you tried this:

```
100 IF A=B THEN PRINT"A=B":GOTO 2000:
PRINT"NO MATCH"
```

PET will never, ever find PRINT "NO MATCH." If A=B, it will go to 2000 before it gets to the PRINT "NO MATCH" -- and if A does not equal B, PET will go on to the next numbered line.

This code takes up 70 bytes of memory -- and it works:

```
500 FOR I=1 TO 10
510 IF A(I)=B(I) THEN 540
520 PRINT"NO MATCH"
530 NEXT I
540 PRINT"FOUND IT"
```

And, to save space, we concatenate:

```
500 FOR I=1 TO 10:IF A(I)=B(I) THEN 540:PRINT
"NO MATCH":NEXT I
540 PRINT"FOUND IT"
```

which takes 60 bytes of memory -- and doesn't work. PET will never print "NO MATCH" and it will never find the NEXT I part. Because if A(I)=B(I), PET will jump out of the loop to line 540. And, if A(I) does not equal B(I), PET will stop scanning line 500 and will fall through to line 540. A safer way to do it is:

```
500 FOR I=1 TO 10:IF A(I)=B(I) THEN 540
520 PRINT"NO MATCH":NEXT I
540 PRINT"FOUND IT"
```

and this works -- and takes up 62 bytes of memory.

If you want to try all these examples, remember to assign values to the arrays A and B before you start:

with this program:

```
10 PRINT"Ⓕ◯◎ Ⓛ";:FOR I=1 TO 100:NEXT I
20 PRINT"◯ Ⓛ";:FOR I=1 TO 100:NEXT I
30 GET A$:IF A$="" THEN 10
```

And now you have a blinking cursor! You can add a line 40 PRINT A$ if you want to be sure PET picked up the character you typed.

GET can be useful when the programmer expects a simple "YES/NO" response from the user of his program. PET will pick up the Y or the N and go on, so the user doesn't have to type in the entire work. GET can be useful when a single character is needed in a real-time game situation (such as the LUNAR LANDER demo seen at some Commodore exhibits), and GET can be useful when trying to draw pictures under program control.

If the programmer wants to input longer strings, he can try this code, added to the above program:

```
40 IF A$=CHR$(13) OR A$=CHR$(141) THEN 80
50 PRINT A$;
60 B$=B$+A$
70 GOTO 10
80 PRINT:PRINT B$
90 PRINT"DONE"
```

In line 40, the code values 13 and 141 represent the unshifted and the shifted RETURN. Line 50 prints A$. The programmer should be aware that because of the timing loops (the FOR/NEXT loops) in lines 10 and 20, the user may be able to type faster than PET prints the character. This can be changed, if desired, by decreasing the number of times through the loops (change 100 to 50). Line 60 puts all the A$ characters together into one long string, and line 70 tells PET to check for another input character.

The following routine allows the user to read a screenful of data (or just a few lines at a time), then signal PET by pressing a key that he is ready to receive more information.

```
3000 *** REM WAIT ROUTINE ***
3010 PRINT"Ⓗ"                SEND THE CURSOR HOME
3020 FOR I=1 TO 22
3030 PRINT"Ⓓ"                MOVE TO BOTTOM OF SCREEN
3040 NEXT I
3050 PRINT"PRESS ANY KEY TO CONTINUE"
3060 GETA$:IF A$="" THEN 3060
```

The next line should be a RETURN, if you use this routine as a subroutine called after each screenful of data is printed.

Now you know all about the GET keyword (it says here!). If we've left anything out that you think should have been included, let us know and we'll print that information, too.

LINE-NUMBERS: When you have many things to do, you may make a list of them to be sure you don't forget anything. Your list may look like this:

1. Get stamps.
2. See about cost for fixing TV.
3. Call J.B. re: car insurance.
4. Pick up dry cleaning.
5. Call R.P. re: Saturday's dance.
6. Order new address labels.

You may not do all these things in exactly this order, but that's because you can go over the list again and again and check things off as you get them done.

Your PET needs a list of things to do, too, or it will forget everything you tell it. The list is called a "program." Each item on a list has to have a number, so PET will know the order in which you want it to do things, since PET can't look back over the list.

Line numbers are very important to PET. First, without line numbers, PET will forget your commands as soon as it finishes executing them, so you'll have to type the command every time you want PET to perform. Second, PET can't decide in what order you want it to execute the commands. And PET can't even guess. So if you have more than one thing for PET to do, you have to have line numbers to let it know which thing to do first and which to do next. And, finally, line numbers make it possible for you to change the order in which PET is supposed to do things, by telling PET to go to one line if an expression* is true and to a different line if the expression is false.

Over the years, as BASIC has become more and more popular, programmers have learned that if they use line numbers in increments of 10 (i.e., 10, 20, 30, 40, etc.), they don't have to retype the whole program if they forget something which should have been in the middle of the list. If you have line numbers 1, 2, 3, 4, 5, there's no way to insert a new line between lines 2 and 3. If you use line numbers 10, 20, 30, 40, 50, you can insert 9 lines more, if you have to, between lines 20 and 30. So -- use line numbers in increments of 10. In fact, that's so important, you should put it on YOUR list of things to remember!

Another neat thing about the way PET handles line numbers is that you don't have to type the lines in the same order as you want PET to execute them. You just have to NUMBER the lines correctly. For instance, if you have a program with lines 10, 20, 30 and 40 already typed in, you don't have to go through any strenuous exercises to get PET to accept a line number 25. You can just type it at the end of your program and PET will put it in the right place in the program.

Not only that, but if you typed an error in line 30 (and don't find it until you're trying to RUN the program) you can just retype the line. PET will delete the old line 30 and replace it with the new one.

Furthermore, if you decide you don't need a specific line after all, you can get rid of it by just typing the number and pressing the RETURN key. PET will delete that line from your program.

Remember that PET does allow you other means of dealing with

typographic errors, but we'd really get sidetracked if we go into that now.

LIST: Now that you understand line numbers, LIST begins to make sense. When you tell PET to LIST, and press the return key, PET lists all the numbered lines in your program. Suppose you have 10 lines -- lines 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. If you type LIST, PET lists all the lines. If you type LIST20, PET lists line 20. If you type LIST30-, PET lists all the lines from line 30 to line 100 (the end of the program): 30, 40, 50, 60, 70, 80, 90, 100. If you type LIST40-70, PET lists lines 40, 50, 60, 70. If you type LIST-80, PET lists all the lines up to and including line 80: 10, 20, 30, 40, 50, 60, 70, 80.

The secret is in the minus sign, of course. Let's summarize, using L to mean "line number," and N to indicate a second line number when needed:

LIST	List all program lines.
LIST-L	List all lines up to and including line L.
LIST L	List only line L.
LISTL-N	List lines L through line N.
LISTL-	List line L and all lines following it.

RUN: RUN tells PET to start doing all the things you told it to do in your program. PET is ready to start from the very beginning, knowing nothing except that it has a list of things to do.

You can tell PET to run only certain segments of your program. RUN50 tells PET to RUN the program, but to start at line 50 instead of at the very beginning of the program.

Now you know the MOST IMPORTANT things -- how to LIST and RUN your programs. (When you want to type in a new program, tell PET NEW -- otherwise it will add your new program to the old one. If you have an old program which uses lines 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, and you type a new program which uses lines 10, 20, 30,40, 50, what you'll get is a new program with the new lines 10, 20, 30, 40, 50, AND lines 60, 70, 80, 90, 100 from the old program. Be sure to type NEW before entering any new program!)

Now let's look at some KEYWORDS. Keywords tell PET what it's supposed to do. (Remember, line numbers only tell PET the order of things to do. Keywords tell PET what things it's being asked to perform.)

DATA statements tell PET what information it can use while running your program. If you put your data in DATA statements, then PET won't forget it every time you type RUN. But each item in a DATA statement can only be used once. You have to put the data in the DATA statement in the same order in which you want PET to use the data, from left to right. (DATA statements can contain alphabetic or graphic characters, of course, but we'll only talk about numeric data this time.)

READ statements tell PET to read the data contained in the data statement. Like people, PET reads from left to right.

RESTORE tells PET to start all over at the beginning of the DATA

```
3000 REM ** X$, Y$ CURSOR CONTROL
3050 PRINT LEFT$(Y$,Y);LEFT$(X$,X);A$:RETURN
```

GET

The GET keyword permits input of one character at a time from the PET keyboard. The greatest value to the user is that GET eliminates the need to use the carriage return after typing in data. The greatest drawback is that GET is so fast, the user doesn't have time to type a character before PET goes on with the rest of the program. Characters input with the GET keyword are not printed on the CRT, either, so the user has to make allowances for that by adding a PRINT statement to his code.

```
10 GET A$
20 PRINT A$
30 PRINT"DONE"
```

Run the above program and you'll see what we mean. Note that PET did not wait for you to type a character before printing the DONE message. A "null" or empty string is considered a valid character, and since you didn't type fast enough, PET assigned a null string to A$, then moved on.

```
10 GET A$:IF A$=""THEN 10
20 PRINT A$
30 PRINT"DONE"
```

In this program, PET is instructed to continue trying to GET a character from the keyboard if A$ contains a null string. PET will sit at line 10 and wait until the user presses a key (any key except STOP). If the user presses the RETURN key, PET will store the code for RETURN in A$, but will not jump out of the program as it does when the INPUT keyword is used. The only way out of a GET statement is through the STOP key.

```
10 GET A$:IF A$=""THEN 10
20 IF A$=" ○ " THEN 50
30 PRINT A$
40 PRINT"DONE":END
50 PRINT"PRESS THE SPACE KEY":GOTO 10
```

In this program, PET waits at line 10 until the user presses a key. Line 20 tells PET that if A$ does not contain a space or blank character, it should go to line 50. Line 40 tells PET that if A$ contains anything other than a space, it should print the message "DONE" and end the program. Line 50 contains an instruction message for the user, and tells PET to go back to line 10 and let the user try again.

Have you noticed that PET does not supply a question mark or a blinking cursor when GET is used instead of INPUT? You can fix that

USEFUL PET ROUTINES

-- or: Why Re-invent The Wheel?

This column will be devoted to those routines we are all apt to be using lots of times. Such things as a bubble sort routine, a routine to force PET to wait for a character when you're using the GET keyboard, and so on. Remember that if you use these routines as subroutines, you must add a RETURN statement at the end.

```
3000 REM ** FRAME ROUTINE ***
3010 PRINT"(S)":FOR I=1TO39:PRINTF$;:NEXTI:PRINT"(U)"
3020 FOR I=1TO23:PRINTTAB(0)F$TAB(38)F$:NEXTI
3030 FOR I=1TO39:PRINTF$;:NEXTI:PRINT"(H)"
```

In this routine, F$ can represent any one of the PET's unique graphic characters. You can replace F$ in the routine with your favorite character (enclosed in quotes, of course) or change F$ every time you draw the frame. For example,

```
40 F$="X":GOSUB3000     or     140 F$="[graphic character]":GOSUB3000
```

But remember -- if you use this routine as a subroutine, you must add a line 3040 RETURN, and you must have an END statement somewhere above the routine so you don't fall into and through it as your program executes.

X$,Y$ CURSOR CONTROL

The programmability of PET's cursor controls permits the user to place the cursor anywhere on the CRT with a minimum of effort.

At the beginning of the main program the user should assign values to X$, Y$, X, Y, and A$, so PET doesn't have to reassign them each time through a subroutine. The assignments are placed in the routine here so it will act as a stand-alone routine. In real life, the routine itself can consist of a single statement.

```
3000 REM *** X$, Y$ CURSOR CONTROL
3010 X$="(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)
(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)(R)"
3020 Y$="(H)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)(D)
(D)(D)(D)(D)(D)(D)(D)"
3030 X=10:Y=10:A$="HELLO":PRINT"(S)"
3040 PRINT LEFT$(Y$,Y);LEFT$(X$,X);A$
3050 END
```

Statements 3010, 3020 should be included at the beginning of the main program (in the first few lines of code). Statement 3030 has the X (number of spaces right), Y (number of lines down) and print message (A$); these can be changed anywhere in the main program before a GOSUB3000. Then lines 3040 and 3050 can be combined -- and your routine becomes:

statement. Since each data item can normally be used only once, RESTORE tells PET it has permission to start at the beginning so each item can be used a second time.

PRINT tells PET to display something on the screen.

So let's write a program. Something to read and print two numbers, so you can concentrate on the programming instead of the problem:

```
10 DATA 10,20          This is the data.
20 READ A,B            PET reads it
30 PRINT A,B           And prints it
40 RESTORE             And restores the data
50 READ C,D            To read it
60 PRINT C,D           And print it again.
```

Notice all those commas? Every item in a list of items has to be separated from all the other items in the list. The commas do the separating. What are all the unexplained letters (A, B, C, D)? PET has to save the data in storage places, and the storage places have to have labels so PET can remember where it stored stuff. Letters of the alphabet are used to label the storage places. (We could get into a very complicated explanation about "memory locations," "variables," "variable names," and so on, but it's not necessary. Buy a BASIC book if you really want to know all that. The important thing is that you understand that PET stores stuff in places which need labels so PET can find the stuff again. Any letter of the alphabet will do, for starters.)

Okay. Type the program and RUN it. Try deleting line 40 (type 40 and hit RETURN) and RUN the program again, to understand why RESTORE is so important. PET tells you it's out of data, right? So retype line 40 and you're all set.

So what good is all this? Not much. But PET has a neat edit feature so we can change the lines. (If you don't have a copy of Commodore's software bulletin, PET EDITING, call them (415-326-4000-X203) and ask for one. Meantime, just retype the line in order to change it. PET will replace the old line with the retyped line if they both have the same line number.) Now, retype line 30. Just change the comma in line 30 to a + (so line 30 says 30 PRINT A+B) and change the comma in line 60 to a / on the numeric keypad (so line 60 says 60 PRINT A/B). Now RUN. PET added in line 30 and divided in line 60. Change line 30 (+ to a -), and line 60 (/ to an *). Now PET subtracts in line 30 and multiplies in line 60. RUN it. Change line 10 to 10 DATA 50,100 and start over with commas, + and /, and - an *. Or change the DATA items to suit yourself!

Let's do a more complicated program which uses a new keyword, GOTO. GOTO tells PET just that -- GO TO someplace else.

```
10 DATA 10,20,30,40,50,60,70,80
20 READ A,B
30 PRINT A*B,A/B,A+B,A-B
40 GOTO 20
```

Now RUN this.

Now, when PET runs out of data it will stop and print an error message. But meantime, it has printed four sets of numbers on the

screen. With the addition of the IF keyword, we can get away from the error message.

```
10 DATA 10,20,30,40,50,60,70,80
20 READ A,B
30 PRINT A*B,A/B,A+B,A-B
40 IF B=80 GOTO 60
50 GOTO 20
60 END
```

Do you see what good the IF statement is? It tells PET to do something IF the specified condition (in this case, B=80) is true. If the condition IS true, PET goes to line 60 and ends the program. If the condition is NOT true, PET executes line 50. Neat, isn't it?

Let's look at what we've covered in this article:

LINE NUMBERS--NUMBERS which tell PET the order in which it is supposed to obey your instructions.

LIST--List tells PET to list the program lines -- to LIST your instructions.

RUN--Tells PET to start doing whatever you specified in your list of instructions, to execute or RUN the program.

DATA--Tells PET what data it can use while following your instructions.

READ--Tells PET to read the data from the DATA statement.

PRINT--Tells PET to display something on the screen.

RESTORE--Tells PET to start over at the beginning of the DATA statement and that it can use the data again.

IF--Tells PET it has to make a decision about whether something is true or false.

GOTO--Tells PET to go to a specific line to get its next instruction instead of going to the next line in the proper order of things.

*,/,+,- --Arithmetic operators, multiply, divide, add, subtract.

And here are the definitions we promised:

Keyword -- A keyword is the word that tells the computer WHAT to do. RUN, LIST, DATA, READ, PRINT, RESTORE, IF, and GOTO are keywords.

Operands -- An operand is what PET does something TO. In our program, A, B, C, and D are operands. Line numbers are operands in the GOTO statements.

Expression -- An expression is something PET can evaluate. The answer always comes out to a number. An expresssion is always on the RIGHT side of an equals sign (or sometimes a not equals or other such sign).

Condition -- The condition is something PET has to decide the truth or non-truth of. Conditions are used in IF statements.

Now that you got this far, write a letter and tell us what you think of the way we presented this material. Is it clear? Easy to understand? Your feedback will help us do our next presentation in the INTRO TO BASIC section of the PET PAPER. Next time, we'll talk more about PRINT and DATA, and we'll cover FOR/NEXT loops and INPUT statements.

USER GROUPS!

We'd like to list your PET User Group in the PET PAPER. We need the name of someone to contact, an address, dures infomation -- all that sort of thing. We want to list all the local groups -- PET owners in your community may not know how to reach you.

We'd also like permission to reprint your Club Notes in the PET PAPER. Authors and Groups will be given byline credit, so everyone will know where to go for more information on the specific topic.

Hopefully, it won't be long before everyone in the country has access to all the PET information, software, and hardware in existence. Let us know what you know. Let us know what you want to know. Let us help the PET Users of the world unite!

PET PROSE

There are many people who want specialized application programs to use on their PETs, but who are not knowledgeable enough to write them. These people are willing to pay for the software they want. If you can write significant software in some specific field (i.e., education, business, engineering, etc.) and are willing to do so, we can help you find the people who are willing to pay you to write programs for them. Send us $25 and we'll print your name, address, phone number, and your field of expertise, in the next 10 issues of the PET PAPER.

If you are someone who WANTS a program written for your specialized purpose, we can help you find the programmer to write it. Just send us your name, address, phone number, and some idea of what you're looking for. We'll print your request for help in the next issue of the PET PAPER -- at no charge to you!