Proxa

commodore
pet computer

USERS HANDBOOK

B A S I C M A N U A L

****

C O N T E N T S

****

****

I N T R O D U C T I O N

Before a computer can perform any useful function, it must be 'told' what to do. Unfortunately, at this time, computers are not capable of understanding English or any other 'human' language. This is primarily because our languages are rich with ambiguities and implied meanings. The computer must be told precise instructions and the exact sequence of operations to be performed in order to accomplish any specific task. Therefore, in order to facilitate human communication with a computer, programming languages have been developed.

PET BASIC is a programming language both easily understood and simple to use. It serves as an excellent 'tool' for applications in areas such as business, science and education. With only a few hours of using BASIC, you will find that you can already write programs with an ease that few other computer languages can duplicate.

Originally developed at Dartmouth University, BASIC language has found wide acceptance in the computer field. Although it is one of the simplest computer languages to use, it is very powerful. BASIC uses a small set of common English words as its 'commands'. Designed specifically as an 'interactive' language, you can give a command such as "PRINT 2 + 2", and BASIC will immediately reply with "4". It is not necessary to submit a card deck with your program on it and then wait hours for the results. Instead, the full power of the PET is 'at your fingertips'.

We hope that you enjoy BASIC and are successful in using it to solve all of your programming needs.

In order to maintain a maximum quality level in our documentation, we will be continuously revising this manual. If you have any suggestions on how we can improve it, please let us know.

If you are already familiar with BASIC programming, the following section may be skipped. Turn directly to the Reference Material on Page 26.

This section is not intended to be a detailed course in BASIC programming. It will, however, serve as an excellent introduction for those of you unfamiliar with the language.

The text here will introduce the primary concepts and uses of BASIC enough to get you started writing programs.

We recommend that you try each example in this section as it is presented. This will enhance your "feel" for BASIC and how it is used.

When PET is turned on, the following should appear:

```
*** COMMODORE BASIC ***

7167 BYTES FREE

READY.
```

NOTE: All commands to PET should end with a 'RETURN'. The 'RETURN' tells BASIC that you have finished typing the command.

Now, try typing in the following:

```
PRINT 10-4 (end with RETURN)
```

PET BASIC will immediately print:

```
6

READY.
```

The print statement you typed in was executed as soon as you hit the 'RETURN' key. BASIC evaluated the formula after the PRINT and then printed its value, in this case 6.

Now try typing in this:

```
PRINT 1/2,3*10    ("*" means multiply, "/" means
                                        divide)
```

PET will print:

```
.5        30
```

As you can see, PET BASIC can do division and multiplication as well as subtraction. Note how a "," (comma) was used in the PRINT command to print two values instead of just one. The comma divides the 40 character line into 4 columns, each 10 characters wide. The result of a "," causes BASIC to skip to the next 10 column field where the value 30 was printed.

Commands such as the PRINT statements you have just typed in are called Direct Commands. There is another type of command

called an Indirect Command. Every Indirect Command begins with a Line Number. A Line Number is any integer from 0 to 64000.

Try typing in the following lines:

```
10 PRINT 2+3
20 PRINT 2-3
```

A sequence of Indirect Commands is called a "Program". Instead of executing indirect statements immediately, BASIC saves Indirect Commands in the memory. When you type in RUN, BASIC will execute the lowest numbered indirect statement that has been typed in first, then the next highest, etc., for as many as were typed in.

Suppose we type in RUN now:

```
RUN
```

BASIC will type out:

```
 5
-1
READY.
```

In the example above, we typed in line 10 first and line 20 second. However, it makes no difference in what order you type in indirect statements. BASIC always puts them into correct numerical order according to the Line Number.

If we want a listing of the complete program currently in memory, we type in LIST. Type this in:

```
LIST
```

BASIC will reply with:

```
10 PRINT 2+3
20 PRINT 2-3
READY.
```

Sometimes it is desirable to delete a line of a program altogether. This is accomplished by typing the Line Number of the line we wish to delete, followed only by a carriage return.

Type in the following:

```
10
LIST
```

BASIC will reply with:

```
20 PRINT 2-3
READY.
```

We have now deleted line 10 from the program. There is no way to get it back. To insert a new line 10, just type in 10 followed by the statement we want BASIC to execute.

Type in the following:

```
10 PRINT 2*3
LIST
```

BASIC will reply with:

```
10 PRINT 2*3
20 PRINT 2-3
READY.
```

There is an easier way to replace line 10 than deleting it and then inserting a new line. You can do this by just typing the new line 10 and hitting the carriage return. BASIC throws away the old line 10 and replaces it with the new one.

Type in the following:

```
10 PRINT 3-3
LIST
```

BASIC will reply with:

```
10 PRINT 3-3
20 PRINT 2-3
READY.
```

It is not recommended that lines be numbered consecutively. It may become necessary to insert a new line between two existing lines. An increment of 10 between line numbers is generally sufficient.

If you want to erase the complete program currently stored in memory, type in NEW. If you are finished running one program and are about to read in a new one, be sure to type in NEW first. This should be done in order to prevent a mixture of the old and new programs.

Type in the following:

```
NEW
```

BASIC will reply with:

```
READY.
```

Now type in:

```
LIST
```

BASIC will reply with:

```
READY.
```

Often it is desirable to include text along with answers that are printed out, in order to explain the meaning of the numbers.

Type in the following:

```
PRINT "ONE THIRD IS EQUAL TO",1/3
```

BASIC will reply with:

```
ONE THIRD IS EQUAL TO          .333333333
READY.
```

As explained earlier, including a "," in a print statement causes it to space over to the next 10 column field before the value following the "," is printed.

If we use a ";" instead of a comma, the value next will be printed immediately following the previous value.

NOTE: Numbers are always printed with at least one trailing space. Any text to be printed is always to be enclosed in double quotes.

Try the following examples:

A)
```
PRINT "ONE THIRD IS EQUAL TO"; 1/3
ONE THIRD IS EQUAL TO .333333333
READY.
```

B)
```
PRINT 1,2,3
1          2          3
READY.
```

C)
```
PRINT 1;2;3
1 2 3
READY.
```

D) PRINT -1;2;-3

```
-1 2 -3
READY.
```

We will digress for a moment to explain the format of numbers in BASIC. Numbers are stored internally to over nine digits of accuracy. When a number is printed, only nine digits are shown. Every number may also have an exponent (a power of ten scaling factor).

The largest number that may be represented in BASIC is $1.7*10^{38}$, while the smallest positive number is $2.93*10^{-39}$.

When a number is printed, the following rules are used to determine the exact format:

1) If the number is negative, a minus sign (-) is printed. If the number is positive, a space is printed.

2) If the absolute value of the number is an integer in the range 0 to 999999999, it is printed as an integer.

3) If the absolute value of the number is greater than or equal to .1 and less than or equal to 999999999, it is printed in fixed point notation, with no exponent.

4) If the number does not fall under categories 2) or 3), scientific notation is used.

Scientific notation is formatted as follows:

SX.XXXXXXXXESTT (each X being some integer 0 to 9)

The leading "S" is the sign of the number, a space for a positive number and a "-" for a negative one. One non-zero digit is printed before the decimal point. This is followed by the decimal point and then the other eight digits of the mantissa. An "E" is then printed (for exponent), followed by the sign "S" of the exponent; then the two digits "TT" of the exponent itself. Leading zeroes are never printed; i.e. the digit before the decimal is never zero. Also, trailing zeroes are never printed. If there is only one digit to print after all trailing zeroes are suppressed, no decimal point is printed. The exponent sign will be "+" for positive and "-" for negative. Two digits of the exponent are always printed; that is zeroes are not suppressed in the exponent field. The value of any number expressed thus is the number to the left of the "E" times 10 raised to the power of the number to the right of the "E".

No matter what format is used, a space is always printed following a number.

The following are examples of various numbers and the output format BASIC will place them into:

NUMBER	OUTPUT FORMAT
+1	1
-1	-1
6523	6523
-23.460	-23.46
1E20	1E+20
-12.3456789E-7	-1.23456789E-06
1.234567E-10	1.234567E-10
1000000	1E+06
999999999	999999999
.1	.1
.01	1E-02
.000123	1.23E-04

A number input from the keyboard or a numeric constant used in a BASIC program may have as many digits as desired, up to the maximum length of a line (72 characters). However, only the first 9 digits are significant, and the ninth digit is rounded up.

```
PRINT 1.2345678901234567890
1.23456789
READY.
```

The following is an example of a program that reads a value from the keyboard and uses that value to calculate and print a result:

```
10 INPUT R
20 PRINT 3.14159*R*R
RUN
?  10
314.159
READY.
```

<u>NOTE:</u> "π" may be used as 3.14159265 at any time. This saves

memory space and decreases execution time considerably.

Here is what is happening. When BASIC encounters the input statement, it types a question mark (?) on the screen and then waits for you to type in a number. When you do (in the above example 10 was typed), execution continues with the next statement in the program after the variable (R) has been set (in this case to 10). In the above example, line 20 would now be executed. When the formula after the PRINT statement is evaluated, the value 10 is substituted for the variable R each time R appears in the formula. Therefore, the formula becomes 3.14159*10*10, or 314.159.

If you have not already guessed, what the program above actually does is to calculate the area of a circle with the radius "R".

If we wanted to calculate the area of various circles, we could keep re-running the program over each time for each successive circle. But there is an easier way to do it simply by adding another line to the program as follows:

```
30 GOTO 10
RUN
? 10
314.159
? 3
28.2743
? 4.7
69.3977
READY.
```

By putting a "GOTO" statement on the end of our program, we have caused it to go back to line 10 after it prints each answer for the successive circles. This could have gone on indefinitely, but we decided to stop after calculating the area for three circles. This was accomplished by typing a RETURN to the input statement (thus a blank line).

The letter "R" in the program we just used was termed a "variable". A variable name can be any alphabetic character and may be followed by any alphanumeric character.

Any alphanumeric characters after the first two are ignored, but accepted. An alphanumeric character is any letter (A-Z) or any number (0-9).

Below are some examples of legal and illegal variable names:

LEGAL	ILLEGAL
A%	(% after a variable means "INTEGER" and uses only two bytes of memory and has the range -32767 to 32767.)
TP	TO (variable names cannot be reserved words).
PSTG$	RGOTO (variable names cannot contain reserved words)
COUNT	

RESERVED VARIABLES

TI and TI$

TI is a 7 digit counter that is incremented every 1/60 second.

TI$ is a string variable which behaves as a 24 hour clock and is formatted "HHMMSS". You may set TI$ in this fashion: TI$ = "HHMMSS".

The words used as BASIC statements are "reserved" for this specific purpose. You cannot use these words as variable names for inside of any variable name. For instance, "FEND" would be illegal becasue "END" is a reserved word.

The following is a list of the reserved words in BASIC:

ABS CLR DATA DIM END FOR GOSUB GOTO IF INPUT
INT LET LIST NEW NEXT PRINT READ REM RESTORE
RETURN RND RUN SGN SIN SQR STEP STOP TAB(
THEN TO USR SYS PRINT# GET CLOSE CMD OPEN ASC
AND ATN CHR$ LOAD CONT COS SAVE DEF EXP FN
FRE LEFT$ LEN LOG MID$ ON OR NOT PEEK POKE
POS RIGHT$ SPC(STR$ TAN VAL WAIT

Besides having values assigned to variables with an input statement, you can also set the value of a variable with a LET or assignment statement.

The "?" character may be used at any time instead of the keyword PRINT.

Try the following examples:

```
A=5

READY.

PRINT A, A*2
 5          10

READY.

LET Z=7

READY.

PRINT Z, Z-A
 7          2

READY.
```

As can be seen from the examples, the "LET" is optional in an assignment statement.

BASIC "remembers" the values that have been assigned to variables using this type of statement. This "remembering" process uses space in the memory to store the data.

The values of variables are thrown away and the space in memory used to store them is released when one of four things occur:

1) A new line is typed into the program or an old line is deleted.

2) A CLR command is typed in

3) A RUN command is typed in

4) NEW is typed in

Another important fact is that if a variable is encountered in a formula before it is assigned a value, it is automatically assigned the value zero. Zero is then substituted as the value of the variable in the particular fomula. Try the example below:

```
PRINT Q,Q+2,Q*2
 0          2          0

READY.
```

Another statement is the REM statement. REM is short for remark. This statement is used to insert comments or notes into a program. When BASIC encounters a REM statement the rest of the line is ignored.

This serves mainly as an aid for the programmer himself, and serves no useful function as far as the operation of the program in solving a particular problem.

Surpose we want to write a program to check if a number is zero or not. With the statements we've gone over so far this could not be done. What is needed is a statement which can be used to conditionally branch to another statement. The "IF-THEN" statement does just that.

Try typing in the following program: (remember, type NEW first)

```
10 INPUT B

20 IF B=0 THEN 50

30 PRINT "NON-ZERO"

40 GOTO 10

50 PRINT "ZERO"

60 GOTO 10
```

When this program is typed into the PET and run, it will ask for a value of B. Type any value you wish in. The program will then come to the "IF" statement. Between the "IF" and the "THEN" portion of the statement there are two expressions separated by a relation.

A relation is one of the following six symbols:

RELATION	MEANING
=	EQUAL TO
>	GREATER THAN
<	LESS THAN
<>	NOT EQUAL TO
<=	LESS THAN OR EQUAL TO
>=	GREATER THAN OR EQUAL TO

The IF statement is either true or false, depending upon whether the two expressions satisfy the relation or not. For example, in the program we just did, if 0 was typed in for B the IF statement would be true because 0=0. In this case, since the number after the THEN is 50, execution of the program would continue at line 50. Therefore, "ZERO" would be printed and then the program would jump back to line 10 (because of the GOTO statement in line 60).

Suppose a 1 was typed in for B. Since 1=0 is false, the IF statement would be false and the program would continue execution with the next line. Therefore, "NON-ZERO" would be printed and the GOTO in line 40 would send the program back to line 10.

Now try the following program for comparing two numbers:

```
10 INPUT A,B
20 IF A<=B THEN 50
30 PRINT "A IS BIGGER"
40 GOTO 10
50 IF A<B THEN 80
60 PRINT "THEY ARE THE SAME"
70 GOTO 10
80 PRINT "B IS BIGGER"
90 GOTO 10
```

When this program is run, line 10 will input two numbers from the terminal. At line 20, if A is greater than B, A<=B will be false. This will cause the next statement to be executed printing "A IS BIGGER" and then line 40 sends the computer back to line 10 to begin again.

At line 20, if A has the same value as B, A<=B is true so we go to line 50. At line 50, since A has the same value as B, A<B is false; therefore, we go to the following statement and print "THEY ARE THE SAME". Then line 70 send us back to the beginning again.

At line 20, if A is smaller than B, A<=B is true so we go to line 50. At line 50, A<B will be true so we then go to line 80. "B IS BIGGER" is then printed and again we go back to the beginning.

Try running the last two programs several times. It may make it easier to understand if you try writing your own program at this time using the IF-THEN statement. Actually trying programs of your own is the quickest and easiest way to understand how BASIC works. Remember, to stop these programs just give a carraige return to the input statement.

One advantage of computers is their ability to perform repetitive tasks. Let's take a closer look and see how this works.

Suppose we want a table of square roots from 1 to 10. The BASIC function for square root is "SQR"; the form being SQR(X), X being the number you wish the square root calculated from. We could write the program as follows:

```
10 PRINT 1,SQR(1)
20 PRINT 2,SQR(2)
30 PRINT 3,SQR(3)
40 PRINT 4,SQR(4)
50 PRINT 5,SQR(5)
60 PRINT 6,SQR(6)
70 PRINT 7,SQR(7)
80 PRINT 8,SQR(8)
90 PRINT 9,SQR(9)
100 PRINT 10,SQR(10)
```

This program will do the job; however, it is terribly inefficient. We can improve the program tremendously by using the IF statement just introduced as follows:

```
10 N=1
20 PRINT N,SQR(N)
30 N=N+1
40 IF N<=10 THEN 20
```

When this program is run, its output will look exactly like that of the 10 statement program above it. Let's look at how it works.

At line 10 we have a LET statement which sets the value of the variable N at 1. At line 20 we print N and the square root of N using its current value. It thus becomes 20 PRINT 1, SQR(1), and this calculation is printed out.

At line 30 we use what will appear at first to be a rather unusual LET statement. Mathematically, the statement N=N+1 is nonsense. However, the important thing to remember is that in a LET statement, the symbol "=" does not signify equality. In this case "=" means "to be replaced with". All the statement does is to take the current value of N and add 1 to it. Thus, after the first time through line 30, N becomes 2.

At line 40, since N now equals 2, N<=10 is true so the THEN portion branches us back to line 20, with N now at a value of 2.

The overall result is that lines 20 through 40 are repeated, each time adding 1 to the value of N. When N finally equals 10 at line 20, the next line will increment it to 11. This results in a false statement at line 40, and since there are no further statements to the program it stops.

This technique is referred to as "looping" or "iteration". Since it is used quite extensively in programming, there are special BASIC statements for using it. We can show these with the following program:

```
10 FOR N=1 TO 10
20 PRINT N,SQR(N)
30 NEXT N
```

The output of the program listed above will be exactly the same as the previous two programs.

At line 10, N is set to equal 1. Line 20 causes the value of N and the square root of N to be printed. At line 30 we see a new type of statement. The "NEXT N" statement causes one to be added to N, and then if N<=10 we go back to the statement following the "FOR" statement. The overall operation then is the same as with the previous program.

Notice that the variable following the "FOR" is exactly the same as the variable after the "NEXT". There is nothing special about the N in this case. Any variable could be used, as long as they are the same in both the "FOR" and "NEXT" statements. For instance, "Z1" could be substituted everywhere there is an "N" in the above program and it would function exactly the same.

Suppose we wanted to print a table of square roots from 10 to 20, only counting by two's. The following program would perform this task:

```
10 N=10
20 PRINT N,SQR(N)
30 N=N+2
40 IF N<=20 THEN 20
```

Note the similar structure between this program and the one listed on page for printing square roots for the numbers 1 to 10. This program can also be written using the "FOR" loop just introduced.

```
10 FOR N=10 TO 20 STEP 2
20 PRINT N,SQR(N)
30 NEXT IN
```

Notice that the only major difference between this program and the previous one using "FOR" loops is the addition of the "STEP 2" clause.

This tells BASIC to add 2 to N each time, instead of 1 as in the previous program. If no "STEP" is given in a "FOR" statement, BASIC assumes that one is to be added each time. The "STEP" can be followed by any expression.

Suppose we wanted to count backwards from 10 to 1. A program for doing this would be as follows:

```
10 I=10
20 PRINT I
30 I=I-I
40 IF I>=1 THEN 20
```

Notice that we are now checking to see that I is <u>greater than</u> or equal to the final value. The reason is that we are now counting by a negative number. In the previous examples it was the opposite, so we were checking for a variable <u>less than</u> or equal to the final value.

The "STEP" statement previously shown can also be used with negative numbers to accomplish this same purpose. This can be done using the same format as in the other program, as follows:

```
10 FOR I=10 to 1 STEP -1
20 PRINT I
30 NEXT I
```

"FOR" loops can also be "nested". An example of this procedure follows:

```
10 FOR I=1 TO 5
20 FOR J=1 TO 3
30 PRINT I,J
40 NEXT I
50 NEXT J
```

It does not work because when the "NEXT I" is encountered, all knowledge of the J-loop is lost. This happens because the J-loop is "inside" of the I-loop.

It is often convenient to be able to select any element in a table of numbers. BASIC allows this to be done through the use of matrices.

A matrix is a table of numbers. The name of this table, called the matrix name, is any legal variable name, "A" for example. The matrix name "A" is distinct and separate from the simple variable "A", and you could use both in the same program.

To select an element of the table, we subscript "A": that is to select the I'th element, we enclose I in parenthesis "(I)" and then follow "A" by this subscript. Therefore, "A(I)" is the I'th element in the matrix "A".

NOTE: In this section of the manual we will be concerned with one-dimensional matrices only. (See Reference Material)

"A(I)" is only one element of matrix A, and BASIC must be told how much space to allocate for the entire matrix.

This is done with a "DIM" statement, using the format "DIM A(15)". In this case, we have reserved space for the matrix index "I" to go from 0 to 15. Matrix subscripts always start at 0; therefore, in the above example, we have allowed for 16 numbers in matrix A.

If "A(I)" is used in a program before it has been dimensioned, BASIC reserves space for 11 elements (0 through 10).

As an example of how matrices are used, try the following to sort a list of 8 numbers with you picking the numbers to be sorted.

```
10 DIM A(8)
20 FOR I=1 TO 8
30 INPUT A(I)
50 NEXT I
70 F=0
80 FOR I=1 TO 7
90 IF A(I)<=A(I+1) THEN 140
100 T=A(I)
110 A(I)= A(I+1)
120 A(I+1)=T
130 F=1
140 NEXT I
150 IF F=1 THEN 70
160 FOR I=1 TO 8
170 PRINT A(I)
180 NEXT I
```

When line 10 is executed, BASIC sets aside space for 9 numeric values, A(0) through A(8). Lines 20 through 50 get the unsorted list from the user. The sorting itself is done by going through the list of numbers and upon finding any two that are not in order, we switch them. "F" is used to indicate if any switches were done. If any were done, line 150 tells BASIC to go back and check some more,

If we did not switch any numbers, or after they are all in order, lines 160 through 180 will print out the sorted list. Note that a subscript can be any expression.

Another useful pair of statements are "GOSUB" and "RETURN" If you have a program that performs the same action in several different places, you could duplicate the same statements for the action in each place within the program.

The "GOSUB"-"RETURN" statements can be used to avoid this duplication.When a GOSUB is encountered,BASIC branches to the line whose number follows GOSUB.However,BASIC remembers where it was in the program before it branched. When the RETURN statement is encountered,BASIC goes back to the first statement following the last GOSUB that was executed. Observe the following program:-

```
 10 PRINT "WHAT IS THE NUMBER";
 30 GOSUB 100
 40 T=N
 50 PRINT "WHAT IS THE SECOND NUMBER";
 70 GOSUB 100
 80 PRINT "THE SUM OF THE TWO NUMBERS IS", T+N
 90 STOP
100 INPUT N
110 IF N = INT(N) THEN 140
120 PRINT "SORRY, NUMBER MUST BE AN INTEGER.  TRY AGAIN".
130 GOTO 100
140 RETURN
```

What this program does is to ask for two numbers which must be integers, and then prints the sum of the two. The subroutine in this program is lines 100 to 130. The subroutine asks for a number, and if it is not an integer, asks for a number again. It will continue to ask until an integer value is typed in.

The main program prints "WHAT IS THE NUMBER", and then calls the subroutine to get the value of the number into N. When the subroutine returns (to line 40), the value input is saved in the variable T. This is done so that when the subroutine is called a second time, the value of the first number will not be lost.

"WHAT IS THE SECOND NUMBER" is then printed, and the second

value is entered when the subroutine is again called.

When the subroutine returns the second time, "THE SUM OF THE TWO NUMBERS IS" is printed, followed by the value of their sum. T contains the value of the first number that was entered and N contains the value of the second number.

The next statement in the program is a "STOP" statement. This causes the program to stop execution at line 90. If the "STOP" statement was not included in the program, we would 'fall into' the subroutine at line 100. This is undesirable because we would be asked to input another number. If we did, the subroutine would try to return; and since there was no "GOSUB" which called the subroutine, an RG error would occur. Each "GOSUB" executed in a program should have a matchine RETURN executed later, and the opposite applies, i.e. a RETURN should be encountered only if it is part of a subroutine which has been called by a GOSUB.

Either "STOP" or "END" can be used to separate a program from its subroutines. "STOP" will print a message saying at what line the STOP was encountered.

Suppose you had to enter numbers to your program that did not change each time the program was run, but you would like it to be easy to change them if necessary. BASIC contains special statements for this purpose, called the READ and DATA statements.

Consider the following program:

```
 10 PRINT "GUESS A NUMBER"
 20 INPUT G
 30 READ D
 40 IF D=-999999 THEN 90
 50 IF D<>G THEN 30
 60 PRINT "YOU ARE CORRECT"
 70 END
 90 PRINT "BAD GUESS, TRY AGAIN"
 95 RESTORE
100 GOTO 10
110 DATA 1,393, -39, 391, -8, 0, 3.14, 90
120 DATA 89, 5, 10, 15, -34, -999999
```

This is waht happens when this program is run. When the READ statement is encountered, the effect is the same as an INPUT statement. But instead of getting a number from the terminal, a number is read from the DATA statements.

The first time a number is needed for a READ, the first number in the first DATA statement is returned. The second time one is needed, the second number in the first DATA statement is returned. When the entire contents of the first DATA statement

have been read in this manner, the second DATA statement will then be used. DATA is always read sequentially in this manner, and there may be any number of DATA statements in your program.

The purpose of this program is to play a little game in which you try to guess one of the numbers contained in the DATA statements. For each guess that is typed in, we read through all the numbers in the DATA statements until we find one that matches the guess.

If more values are read than there are numbers in the DATA statements, an out of data (OD) error occurs. That is why in line 40 we check to see if -999999 was read. This is not one of the numbers to be matched, but is used as a flag to indicate that all the data (possible correct guesses) has been read. Therefore, if -999999 was read, we know that the guess given was incorrect.

Before going back to line 10 for another guess, we need to make the READ begin with the first piece of data again. This is the function of the RESTORE. After the RESTORE is encountered, the next piece of data read will be the first piece in the first DATA statement again.

DATA statements may be placed anywhere within the program. Only READ statements make use of the DATA statements in a program, and any other time they are encountered during program execution they will be ignored.

A list of characters is referred to as a 'String'. PET, COMMODORE and THIS IS A TEST are all strings. Like numeric variables, string variables can be assigned specific values. String variables are distinguished from numeric variables by a "$" after the variable name.

For example, try the following:

```
A$ = "PET 2001"
READY.
PRINT A$
PET 2001
READY.
```

In this example, we set the string variable A$ to the string value PET 2001. Note that we also enclosed the character string to be assigned to A$ in quotes.

Now that we have set A$ to a string value, we can find out

what the length of this value is (the number of characters it contains). We do this as follows:

```
PRINT LEN (A$), LEN ("PET")
8          3
READY.
```

The LEN function returns an integer equal to the number of characters in a string.

The number of characters in a string expression may range from 0 to 255. A string which contains 0 characters is called the "NULL" string. Before a string variable is set to a value in the program, it is initialized to the null string. Printing a null string on the screen will cause no characters to be printed, and the cursor will not be advanced to the next column. Try the following:

```
PRINT LEN (Q$); Q$; 3
0  3
READY.
```

Another way to create the null string is: Q$ = ""

Setting a string variable to the null string can be used to free up the string space used by a non-null string variable.

Often it is desirable to access parts of a string and manipulate them. Now that we have set A$ to "PET 2ØØ1", we might want to print out only the first three characters of A$. We would do so like this:

```
PRINT LEFT$ (A$,3)
PET
READY.
```

LEFT$ is a string function which returns a string composed of the leftmost N characters of its string argument. Here is another example:

```
FOR N=1 TO LEN (A$): PRINT LEFT$ (A$, N): NEXT N
P
PE
PET
PET
PET 2
```

```
PET 2Ø
PET 2ØØ
PET 2ØØ1

READY.
```

Since A$ has 8 characters, this loop will be executed with N=1, 2, 3, . . . 8. The first time through only the first character will be printed, the second time the first two characters will be printed etc.

There is another string function called RIGHT$, which returns the right N characters from a string expression. Try substituting RIGHT$ for LEFT$ in the previous example and see what happens.

There is also a string function which allows us to take characters from the middle of a string. Try the following:

```
FOR N=1 TO LEN (A$): PRINT MID$ (A$, N): NEXT N

PET 2ØØ1
ET 2ØØ1
T 2ØØ1
 2ØØ1
2ØØ1
ØØ1
Ø1
I

READY.
```

MID$ returns a string starting at the Nth position of A$ to the end (last character) of A$. The first position of the string is position 1 and the last possible position of a string is position 255.

Very often it is desirable to extract only the Nth character from a string. This can be done by calling MID$ with three arguments. The third argument specifies the number of characters to return.

For example:

```
FOR N=1 TO LEN (A$): PRINT MID$ (A$, N, 1), MID$
 (A$, N, 2): NEXT N

P          PE
E          ET
T          T
            2
2          2Ø
Ø          ØØ
Ø          Ø1
1          1

READY.
```

See the Reference Material for more details on the workings of LEFT$, RIGHT$ and MID$.

Strings may also be concatenated (put or joined together) through the use of the "+" operator. Try the following:

```
B$ = "UK" + " " + A$
READY.
PRINT B$
UK PET 2ØØ1
READY.
```

Concatenation is especially useful if you wish to take a string apart and then put it back together with slight modifications. For instance:

```
C$ = LEFT$(B$,2)+"-"+MID$(A$,1,3)+"-"+RIGHT$(A$,4)
READY.
PRINT C$
UK-PET-2ØØ1
READY.
```

Sometimes it is desirable to convert a number to its string representation and vice-versa. VAL and STR$ perform these functions.

Try the following:

```
STRING$ = "567.8"
READY.
PRINT VAL(STRING$)
567.8
READY.
STRING$ = STR$(3.1415)
READY.
PRINT STRING$, LEFT$(STRING$,5)
3.1415          3.14
READY.
```

STR$ can be used to perform formatted I/O on numbers. You can convert a number to a string and then use LEFT$, RIGHT$, MID$ and concatenation to reformat the number as desired.

STR$ can also be used to conveniently find out how many print columns a number will take. For example:

```
PRINT LEN(STR$(3.157))
6
READY.
```

If you have an application where a user is typing in a question such as "WHAT IS THE VOLUME OF A CYLINDER OF RADIUS 5.36 FEET, OF HEIGHT 5.1 FEET?", you can use VAL to extract the numeric values 5.36 and 5.1 from the question. For further functions CHR$ and ASC, see Appendix .

The following program sorts a list of string data and prints out the sorted list. This program is very similar to the one given earlier for sorting a numeric list.

```
100 DIM A$(15): REM ALLOCATE SPACE FOR STRING MATRIX
110 FOR I=1 TO 15: READ A$(1): NEXT 1: REM READ IN STRINGS
120 F=0: I=1: REM SET EXCHANGE FLAG TO ZERO AND SUBSCRIPT TO 1
130 IF A$(1)< =A$(I+1) THEN 180: REM DON'T EXCHANGE IF ELEMENTS
    IN ORDER
140 T$=A$(I+1): REM USE T$ TO SAVE A$(I+1)
150 A$(I+1)=A$(I): REM EXCHANGE TWO CONSECUTIVE ELEMENTS
160 A$(I)=T$
170 F=1: REM FLAG THAT WE EXCHANGED TWO ELEMENTS
180 I=I+1: IF I<15 GOTO 130
185 REM ONCE WE HAVE MADE A PASS THRU ALL ELEMENTS, CHECK
187 REM TO SEE IF WE EXCHANGED ANY.  IF NOT, DONE SORTING
190 IF F THEN 120: REM EQUIVALENT TO IF F<>0 THEN 120
200 FOR I=1 TO 15: PRINT A$(I): NEXT I: REM PRINT SORTED LIST
210 REM STRING DATA FOLLOWS
220 DATA APPLE,DOG,CAT,PET,DEREK,KIT
230 DATA MONDAY,"***ANSWER***","FOO"
240 DATA COMPUTER, ABC,LONDON,CAMBRIDGE,LIVERPOOL,ALBUQUERQUE
```

E D I T I N G

The DEL key, when pressed, causes the character to the left of the cursor to be deleted. If this character is not the last character on a line, the line will be shortened accordingly.

The INST (insert) key inserts a space to the left of the cursor and will open out the current line accordingly. If too many spaces are inserted in a line press DEL twice the number of times you have excess spaces. When you are happy with the edited line press RETURN

Use "LIST" nn... to recall the line to be edited from memory.

R E F E R E N C E S E C T I O N

****

COMMANDS

A command is usually given after BASIC has typed READY. This is called the 'Command Level'. Commands may be used as program statements. Certain commands, such as LIST, NEW and LOAD will terminate program execution when they finish.

NAME	EXAMPLE	PURPOSE/USE
CLR	CLR	Clears all variables, resets 'FOR' and 'GOSUB' pointers, RESTOREs data.
LIST	LIST	Lists current program.
	LIST 100-	Optionally starting at specified line. List can be STOPped using the STOP key. (BASIC will finish listing the current line).
	LIST X	Lists just line X.
	LIST -X	Lists from start of program up to line X.
	LIST X -Y	Lists lines X to Y inclusive.
		NOTE: If during listing the RVS key is held down, listing will slow to approximately 2 lines per second.
RUN	RUN	Starts execution of the program currently in memory at the lowest numbered statement. Run deletes all variables (does a CLR) and restores DATA. If you have stopped your program and wish to continue execut-
	RUN 80	ion at some point in the program, use RUN followed by line number.
NEW	NEW	Deletes current program and all variables.
CONT	CONT	Continues program execution after the STOP key is pressed or a STOP statement is executed. You cannot continue after any error, after modifying your program, or before your program has been run. One of the main purposes of CONT is

debugging. Suppose at some point after running your program nothing is printed. This may be because your program is performing some time consuming calculation, but it may be because you have fallen into an 'infinite loop'. An infinite loop is a series of BASIC statements from which there is no escape. The PET will keep executing the series of statements over and over, until you intervene or until power to the PET is cut off. If you suspect your program is in an infinite loop, press STOP, the line number of the statement BASIC was executing will be typed out. After BASIC has typed out READY., you can use PRINT to type out some of the values of your variables. After examining these values, you may become satisfied that your program is functioning correctly. You should then type in CONT to continue executing your program where it left off, or type a direct GOTO statement to resume execution of the program at a different line. You could also use assignment (LET) statements to set some of your variables to different values. Remember, if you press STOP in a program and expect to continue it later, you must not get any errors or type in any new program lines. If you do, you will not be able to continue and will get a cannot continue error. It is impossible to continue a direct commend. CONT always resumes execution at the next statement to be executed in your program when STOP was pressed.

OPERATORS

SYMBOL	SAMPLE STATEMENT	PURPOSE/USE
=	A = 100	Assigns a value to a variable.
	LET Z = 2.5	The LET is optional

-	B=-A	Negation. Note that 0-A is subtraction, while -A is negation.
	130 PRINT X↑3	Exponentiation (equal to X*X*X in the sample statement 0 0 =1 0 to any other power = 0
*	140 X =R*(B*D)	Multiplication
/	150 PRINT X/1.3	Division
+	160 Z=R+T+Q	Addition
-	170 J=100-I	Subtraction

RULES FOR EVALUATING EXPRESSIONS:

1) Operations of higher precedence are performed before operations of lower precedence. This means the multiplication and divisions are performed before additions and subtractions. As an example, 2+10/5 equals 4, not 2.4. When operations of equal precedence are found in a formula, the left hand one is executed first: 6-3+5=8, not -2.

2) The order in which operations are performed can always be specified explicitly through the use of parentheses. For instance, to add 5 to 3 and then divide that by 4, we would use (5+3)/4 which equals 2. If instead we had used 5+3/4, we would get 5.75 as a result (5 plus 3/4).

The precedence of operators used in evaluating expressions is as follows, in order beginning with the highest precedence:
(note: Operators listed on the same line have the same precedence.)

1) Formulas enclosed in parenthesis are always evaluated first
2) ↑ Exponentiation
3) Negation -X Where X may be a formula
4) * / Multiplication and division
5) + - Addition and subtraction
6) Relational operators: =Equal
<> Not equal
< Less Than
> Greater Than
<= Less than or equal
>= Greater than or equal

(These 3 below are logical operators)

7) NOT Logical and bitwise "NOT" like negation, not takes only the formula to its right as an argument
8) AND Logical and Bitwise "AND"
9) OR Logical and Bitwise "OR"

Relational operator expressions will always have a value of True (-1) or a value of False (0). Therefore, (5=4)=0, (5=5)=-1, (4>5)=0, (4<5)=-1 etc. Any value other than zero is taken as TRUE.

The THEN clause of an IF statement is executed whenever the formula after the IF is not equal to 0. That is to say, IF X THEN.... is equivalent to IFX<>0 THEN...

SYMBOL	SAMPLE STATEMENT	PURPOSE/USE
=	10 IF A=15 THEN 40	Expression Equals Expression
<>	70 IF A<>0 Then 5	Expression Does Not Equal Expression
>	30 IFB>100 Then 8	Expression Greater Than Expression.
<	160 If B<2Then 10	Expression less than Expression
<=, =<	180 If 100<=B+C Then100	Expression less than or Equal to Expression.
>=, =>	190 IF Q=>R Then 50	Expression Greater than or equal to Expression.
AND	2 IF A<5 AND B<2 THEN 7	If expression 1 (A<5) AND expression 2 (B<2) are both true, then branch to line 7
OR	IF A<1 or B<2 THEN 2	If either expression 1 (A<1) OR expression 2 (B<2) is true, then branch to line 2
NOT	IF NOT Q3 THEN 4	If expression "NOT Q3" is true because Q3 is false) then branch to line 4 Note:(NOT true = false)

AND, OR and NOT can be used for bit manipulation , and for performing boolean operations.

These three operators convert their arguments to sixteen bit, signed two's complement integers in the range -32768 to +32767 They then perform the specified logical operation on them and return a result within the same range. If the arguments are not in this range, an error results.

The Operations are performed in bitwise fashion, this means that each bit of the result is obtained by examining the bit in the same position for each argument.

The following truth table shows the logical relationship between bits:

OPERATOR	ARG.1	ARG.2	RESULT
AND	1	1	1
	0	1	0
	1	0	0
	0	0	0

OPERATOR	ARG.1	ARG.2	RESULT
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
NOT	1	-	0
	0	-	1

EXAMPLES: (In all of the examples below, leading zeroes on binary numbers are not shown.)

63 AND 16=16	Since 63 equals binary 111111 and 16 equals binary 10000, the result of the AND is binary10000 or 16.
15 AND 14=14	15 equals binary 1111 and 14 equals binary 1110 so 15 and 14 equals binary 1110 or 14.
-1 AND 8=8	-1 equals binary 1111111111111111 and 8 equals binary 1000, so the result is binary 1000 or 8 decimal.
4 AND 2=0	4 equals binary 100 and 2 equals binary 10, so the result is binary 0 because none of the bits in either argument match to give a 1 bit in the result.
4 OR 2=6	Binary 100 OR'd with binary 10 equals binary 110,or 6 decimal.
10 OR10=10	Binary 1010 OR'd with binary 1010 equals binary 1010, or 10 decimal.
-1 OR-2=-1	Binary 1111111111111111 (-1) OR'd with binary 1111111111111110 (-2) equals binary 1111111111111111, or -1.
NOT 0=-1	The bit complement of binary 0 to 16 places is sixteen ones (1111111111111111) or -1. Also Not-1=0
NOT X	NOT X is equal to -(X+1). This is because to form the sixteen bit two's complement of the number, you take the bit (one's) complement and add one.
NOT 1=-2	The sixteen bit complement of 1 is 1111111111111110 which is equal to -(1+1) or -2.

A typical use of the bitwise operators is to test bits set in the computer's I/O locations which reflet the state of some external device. Bit position 7 is the most significant bit of a byte, while position 0 is the least significant.

For instance, suppose bit 1 of location 5000 is 0 when the door to Room X is closed, and 1 if the door is open. The following program will print "Intruder Alert" if the door is opened:

```
10 IF NOT (PEEK(5000)AND 2) THEN 10
```

This line will execute over and over until bit 1 (masked or selected by the 2) becomes a 1. When that happens, we go to line 20.

```
20 PRINT "INTRUDER ALERT"
```

Line 20 will output "INTRUDER ALERT".

However, we can replace statement 10 with a "Wait" statement, which has exactly the same effect.

```
10 WAIT 5000,2
```

This line delays the execution of the next statement in the program until bit 1 of location 5000 becomes 1,. The wait is much faster than the equivalent IF statement and also takes less bytes of program storage.

Sense switches may also be used as an input device by the function. The program below prints out any changes in the sense switches.

```
10 A =300:REM SET A TO A VALUE THAT WILL FORCE PRINTING
20 J=PEEK (sense switch Location) : IF J=A THEN 20
30 PRINT J;:A=J:GO TO 20
```

The following is another useful way of using relational operators:

```
125 A=-(B>C) *B-(B<=C)*C
```

This Statement will set the variable A to MAX (B,C) = the larger of the two variables B and C.

STATEMENTS

Note: In the following description of statements, an argument of V or W denotes a numeric variable, X denotes a numeric expression, X$ denotes a string expression and an I or J denotes and expression that is truncated to an integer before the statement is executed Truncation means that any fractional part of the number is lost, e.g. 3.9 becomes 3, 4.01 becomes 4.

An expression is a series of variables,operators,function calls and constants which after the operations and function calls are performed using the precedence rules,evaluates to a string or numeric value.

A constant is either a number 2.71 or a string literal "abc"

NAME	EXAMPLE	PURPOSE/USE
CLOSE	10 CLOSE N	CLOSEs logical file N (See cassette file)
DATA	20 DATA 1,-3,.04	Specifies data read from left to right.Information appears in data statements in the same order as it will be read in the program.
	30 DATA "ABC",PET	Strings may be read from data statements.If you want the string to contain leading spaces(blanks) colons(:) or commas(,) you must enclose the string in quotes(") It is impossible to have quote marks within a string.
DEF	40 DEF FNA(V)=B+C+V	The user can define functions like the built in functions (SQR,SIN,TAN etc) through the use of the DEF statement.The name of the function is "FN" followed by any legal variable name,for example:FNX,FNJ7,FNPET.User defined functions are restricted to one line.A function may be defined to any expression,but may only have one argument.In the example,B andC are variables that are used in the program.Executing the DEF statement defines the function.User defined functions can be re-defined by executing another DEF statement for the same function.User defined string functions are not allowed. "V" is called the dummy variable.
	50 Z=FNA(3)	Exexcution of this statement following the above would cause Z to be set to B+C+3, but the value of V would be unchanged.

DIM	80 DIM A(3),B(10)	Allocates space for matrices. All matrix elements are set to zero by the DIM statement.
	75 A(5,5),D$(3,4,4)	Matrices may have more than one dimension.Up to 255 dimensions are allowed,but due to the restriction of 80 characters per line the practical maximum is about 34 dimensions.
	35 DIM Q1(N),Z(2*I)	Matrices can be dimensioned dynamically during program execution.If a matrix is not explicitly dimensioned with a DIM statement,it is assumed to be a single dimensioned matrix of whose single subscript may range from 0 to 10 (eleven elements).If
	20 A(8)=4.2	this statement was encountered before a DIM statement for A was found in the program,it would be as if a DIM(10) had been executed previous to the execution of line 20.All subscripts start at zero,which means that DIM(100) really allocates 101 matrix elements.
END	999 END	Terminates program execution without printing a break message.(see STOP). CONT after an END statement causes execution to resume at the statement after the END statement. END can be used anywhere in the program and is optional.
FOR	20 FOR V=1 TO 9.3 STEP .6	(see NEXT statement) V is set equal to the value of the expression following the = in this case 1.This value is called the initial value.Then the statements between FOR and NEXT are executed.The final value is the value of the expression following the TO.The increment is the value of the expression following STEP.When When the NEXT statement is encountered,the step is added to the variable.
	310 FOR V=1 TO 9.3	If no STEP was specified,it is assumed to be one.If the step is positive

and the new value of the variable is the final value (9.3 in this example), or the step value is negative and the new value of the variable is<= the final value , then the first statement following the FOR statement is executed. Otherwise, the statement following the NEXT statement is executed.
All FOR loops execute the statements between the FOR and the NEXT at least once, even in cases like FOR V=1 TO Ø.

315 FOR V=10 N TO 3.4/Q STEP SQR(R) Note that expressions may be used for the initial, final and step values in a FOR loop. The values of the expressions are computed only once, before the body of the FOR.....NEXT loop is executed.

320 FOR V=9 TO 1 STEP -1 When the statement after the NEXT is executed, the loop variable is never equal to the final value, but is equal to whatever value caused the FOR....NEXT loop to terminate. The statements between the FOR and its corresponding NEXT in both examples above (310 & 320) would be executed 9 times.

330 FOR W=1 TO 10: FOR W=1 TO :NEXT W:NEXT W Error: do not use nested FOR...NEXT loops with the same index variable.
FOR loop nesting is limited only by the available memory.

GET

10 GET C Accepts single character from keyboard.

20 GET C$ Accepts single string character from keyboard.

30 GET#D,C Accepts single character from specified logical file.

40 GET#D,C$ Accepts specified single string character from logical file. (SEE CASSETTE FILE).

GOTO

50 GOTO 100 Branches to the statement specified.

GOSUB

10 GOSUB 910 Branches to the specified statement (910) until a RETURN is encountered; when a branch is then made to the statement after the GOSUB. GOSUB nesting is limited only by the available memory.

IF...GOTO

32 IF X =Y+23.4 GOTO 92 Equivalent to IF... THEN, except that IF...GOTO must be followed by a line number, while IF...THEN can be followed by either a line number or another statement.

IF... THEN

IF X<10 THEN 5 Branches to specified statement if the relation is True.

20 IF X<0 THEN PRINT "X LESS THAN 0" Executes all of the statements on the remainder of the line after the THEN if the relation is True.

25 IF X=5 THEN 50:Z=A WARNING. The "Z=A" will never be executed because if the relation is true, BASIC will branch to line 50.
If the relation is false Basic will proceed to the line after line 25.

26 IF X<0 THEN PRINT "ERROR, X NEGATIVE": GOTO 350
In this example, if X is less than 0, the PRINT statement will be executed and then the GOTO statement will branch to line 350. If the X was 0 or positive, BASIC will proceed to execute the lines after line 26.

INPUT

3 INPUT V,W,W2, AB Requests data from terminal (to be typed in). Each value must be separated from the preceeding value by a comma (,).
The last value typed should be followed by carriage return. A "?" is typed as a prompt character. Only constants may be typed in as a response to an INPUT statement, such as 4.5E-3 or "CAT".
If more data was requested in an INPUT statement than was typed in, a "?" is printed and the rest of the data should be typed in. If more data was typed in than was requested, the extra data will be ignored.
Strings must be input in the same format as they are specified in DATA statements. If Alpha data is input when numer ic is expected or vice versa, the BASIC will respond with ?? "REDO FROM START".

5 INPUT "VALUE"; V Optionally types a prompt string ("VALUE") before requesting data from the terminal. If carriage return is typed to an input statement, BASIC returns to command mode.
Typing CONT after an INPUT command has been interrupted will cause execution to resume at the INPUT statement.

	40 INPUT# D,A	Accepts value of A from logical file D.
	50 INPUT# D,A$	Accepts specified string from logical file D.
	60 INPUT# D,A,A$,B,B$	Accepts specified values and strings from logical file D. Strings do not have to be enclosed in quotes. (SEE CASSETTE FILE).
LET	300 LET W=X	Assigns a value to a variable.
	310 V=5.1	"LET" is optional.
LOAD	10 LOAD	Loads next encountered program or file, on built-in tape unit, into PET's memory.
	20 LOAD "NAME"	Loads program or file NAME into memory from built-in tape unit.
	30 LOAD "NAME", D	Loads specified file NAME from device D. (SEE CASSETTE FILE).
OPEN	10 OPEN A	Opens logical file A for read only from built-in tape unit.
	20 OPEN A,D	Opens logical file A for read only from device D.
	30 OPEN A,D,C	Opens logical file A for command C from device D.
	40 OPEN A,D,C, "NAME"	Opens logical file A on device D. If device D accepts formatted files, file NAME is positioned for command. (SEE CASSETTE FILE).
NEXT	340 NEXT V	Marks the end of a FOR loop.
	345 NEXT	If no variable is given, matches the most recent FOR loop.
	350 NEXT V,W	A single NEXT may be used to match multiple FOR statements. Equivalent to NEXT V:NEXT W.
ON...GOTO	100 ON I GOTO 10,20,30,40	Branches to line indicated by the I'th number after the GOTO. That is: IF I=1, THEN GOTO LINE 10 IF I=2, THEN GOTO LINE 20 IF I=3, THEN GOTO LINE 30 IF I=4, THEN GOTO LINE 40. If I=Ø or I attempts to select a non-existent line (>=5 in this case), the statement after the ON statement is executed. However, if I is 255 or 0, an error message will result. As many line numbers as will fit on a line can follow an ON...GOTO.

```
105 ON SGN(X)+2GOTO 40,50,60
```

This statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero, and to line 60 if it is greater than zero.

ON...GOSUB

```
110 ON I GOSUB 50,60
```

Identical to "ON...GOTO", except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the statement after the ON...GOSUB.

POKE

```
357 POKE I,J
```

The POKE statement stores the byte specified by its second argument (J) into the location given by its first argument (I). The byte to be stored must be =>0 and <=255, or an error will occur. The address (I) must be =>0 and <65535, or an error will result. Careless use of the Poke statement will probably cause you to "poke" BASIC to death. A poke to a non-existent memory location is harmless. One of the main uses of POKE is to pass arguments to machine language subroutines. You could also use PEEK and POKE to write a memmory diagnostic or an assembler in BASIC.

PRINT

```
360 PRINT X,Y;Z
370 PRINT
360 PRINT X,Y;
390 PRINT "VALUE IS";A
400 PRINT A2,B,
```

Prints the value of expressions on the terminal. If the list to be printed out does not end with a comma (,), or a semicolon (;), then a carriage return/ line feed is executed after all the values have been printed. Strings enclosed in quotes (") may also be printed. If a semicolon separates two expressions in the list, their values are printed next to each other.

	410 PRINT MID$(A$,2);	String expressions may be printed.
READ	490 READ V,W	Read data into specified varibles from a DATA statement. The first peice of data read will be the first piece of data read will be the first piece of data listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on. When all of the data have been read from the first DATA statement, the next piece of data to be read will be the first piece listed in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an out of data error.
REM	500 REM NOW SET V=0	Allows the programmer to put comments in his program. REM statements are not executed, but can be branched to. A REM statement is terminated by end of line, but not by a ":".
	505 REM SET V=Ø: V=Ø	In this case the V=Ø will never be executed by BASIC.
	506 V=Ø: REM SET V=Ø	In this case V=Ø will be executed
RESTORE	510 RESTORE	Allows the re-reading of DATA statements. After a RESTORE, the next piece of data read will be the first piece listed in the first DATA statement of the program. The second piece listed in the first DATA statement, and so on as in a normal READ operation.

RETURN	50 RETURN	Cause a subroutine to return to the statement after the most recently executed GOSUB.
STOP	900 STOP	Causes a program to stop execution and to enter command mode.
SYS	120 SYS (64824)	Causes the computer to jump to location 64824 decimal in memory and run in machine code from there. Return to command or Basic will a machine code RTS.
TI TI$	75 PRINT TI 85 TI$ = "HHMMSS"	Line 75 Prints the number of JIFFIES since the machine was turned on or the number of JIFFIES equivalent to the time in TI$. Line 85 sets PET'S internal 24 hour clock to real time. JIFFIES are 1/60 TH of a second
USR	95 USR(X)	Transfers program control to a program whose address is at locations 1 and 2.X is a parameter passed to and from the machine language program. (SEE APPENDIX)
VERIFY	10 VERIFY	VERIFIES most recent program saved on built-in cassette by reading it and comparing it with program still in PET's memory.
	20 VERIFY "NAME"	Verifies specified file NAME saved on built-in cassette by reading it and comparing it with program still in PET's memory.
	30 VERIFY "NAME",D	Verifies specified file NAME saved on device D by reading it and comparing it with program still in PET's memory.
WAIT	805 WAIT I,J,K	This statement reads the status of location I, exclusive OR's K with the status, and then AND's the result with J until a non-zero result is obtained. Execution of the program continues at the statement following the WAIT

statement. If the WAIT statement only has two arguments, K is assumed to be zero. If you are waiting for a bit to become zero, there should be a one in the corresponding position of K. I, J and K must be =>0 and <=255.

INTRINSIC FUNCTIONS

ABS(X)	120 PRINT ABS(X)	Gives the absolute value of the expression X. ABS return X if X<=0, X otherwise.
INT(X)	140 PRINT INT(X)	Returns the largest integer less than or equal to its argument X. For example: INT(.23) =0, INT(7)=7, INT(-.1)=1, INT (-2)= -2, INT(1.1)=1. The following would round X to D decimal places: INT(X*10↑D+.5)/10↑D
RND(X)	170 PRINT RND(X)	Generates a random number between Ø and 1. The argument of random numbers as follows: X<Ø starts a new sequence of random numbers using X. Calling RND with the same X starts the same random number sequence. X=Ø gives the last random number generated. Repeated calls to RND(Ø) will always return the same random number. X>Ø generates a new random number between Ø and 1. Note that (B-A)*RND(1)+A will generate a random number between A & B.
SGN(X)	230 PRINT SGN(X)	Gives 1 if X>Ø, Ø if X=Ø, and -1 if X<Ø.
SIN(X)	190 PRINT SIN(X)	Gives the sine of the expressionX.

		X is interpreted as being in radians. Note: COS (X)=SIN (X+3.14159/2) and that 1 Radian =180/P1 degrees=57.2958 degrees; so that the sine of X degrees = SIN(X/57.2958).
SQR(X)	180 PRINT SQR(X)	Gives the square root of the arguement X. An error will occur if X is less than zero.
TAB (I)	240 PRINT TAB (I)	Spaces to the specified print position (column) on the terminal. May be used only in PRINT statements. Zero is the leftmost column on the terminal, 39 the rightmost. If the carriage is beyond position 1, then no printing is done. I must be =>0 and <=255.
ATN (X)	210 PRINT ATN(X)	Gives the arctangent of the arguement X. The result is returned in radians and ranges from -PI/1 to PI/2. (PI/2 = 1.5708)
COS(X)	200 PRINT COS(X)	Gives the cosine of the expression X. X is interpreted as being in radians.
EXP(X)	150 PRINT EXP(X)	Gives the constant "E" (2.718-28) raised to the power X. (E+X) The maximum arguement that can be passed to EXP without overflow occuring is 88.
FRE(X)	270 PRINT FRE(0)	Gives the number of memory bytes currently unused by BASIC.
LOG(X)	160 PRINT LOG(X)	Gives the natural (Base E) logarithm of its arguement X. To obtain the Base Y logarithm of X use the formula LOG(X)/LOG(Y). Example: the base 10 (common) log of 7 = LOG(7)/LOG(10).

PEEK	356 PRINT PEEK (I)	The PEEK function returns the contents of memory address I. the value returned will be =>Ø and <=255. If Iis>65535 or< Ø, an error will occur. An attmpt to read a non-existent memory address will return on unknown value. (see POKE statement)
SPC(I)	250 PRINT SPC(I)	Prints I space (or blank) characters on the screen. May be used only in a PRINT statement. X must be =>Ø and <=255 or an error will result.
TAN(X)	200 PRINT TAN(X)	Gives the tangent of the expression X. X is interpreted as being in radians.

STRINGS

1) A string may be from Ø to 255 characters in length. All string variables end in a dollar sign ($): for example, A$,B9$,K$, HELLO$

2) String matrices may be dimensioned exactly like numeric matrices. For instance, DIM A$(1Ø,1Ø) creates a string matrix of I21 elements, eleven rows by eleven columns (rows Ø to 1Ø and columns Ø to 1Ø). Each string matrix element is a complete string, which can be up to 255 characters in length.

NAME	EXAMPLE	PURPOSE/USE
DIM	25 DIM A$(10,10)	Allocates space for a pointer and length for each element of a string matrix. No string space is allocated.
LET	27 LET A$="PET"+V$	Assigns the value of a string expression to a string variable Let is optional.
=		String comparison operators. Comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If during the comparison of two strings, the end of one is reached, the shorter string is considered smaller Note

		that "A "is greater than "A" since trailing spaces are significant.
+	30 LET A$=B$+C$	String concatentation. The resulting string must be less than 256 characters in length or an error will will occur.
INPUT	40 INPUT X$	Reads a string from the user's keyboard. String does not have to be quoted; but if not, leading blanks will be ignored and the string will be terminated on a "," or ":" character.
READ	50 READ X$	Reads a string from DATA statements within the program. Strings do not have to be quoted; but if they are not, they are terminated on a "," character or end of line and leading spaces are ignored. See DATA for the format of string data.
PRINT	60 PRINT X$ 70 PRINT "FOO"+A$	Prints the string expression on the screen.
	STRING FUNCIONS	
ASC(X$)	300 PRINT ASC(X$)	Returns the ASCII numeric value of the first character of the string expression X$. See Appendix for an ASCII/number conversion table. An error will occur if X$ is the null string.
CHR$(I)	275 PRINT CHR$(I)	Returns a one character string whose single character is the ASCII equivalent of the value of the argument (I) which must be =>Ø and <=255
LEFT$(X$,I)	310 PRINT LEFT$(X$,I)	Gives the leftmost I characters of the string expression X$. If I<=Ø or >255 an error occurs.

LEN(X$)	200 PRINT LEN(X$)	Girves the length of the string expression X$ in characters (bytes). Non-printing characters and blanks are counted as part of the length.
MID$(X$,I)	330 PRINT MID$(X$,I)	MID$ called with two arguments returns characters from the string expression X$starting at character position I. If I LEN(I$), then MID$ returns a null (zero length) string. If I<=Ø or >255, an error occurs.
MID$(X$,I,J)	340 PRINT MID$(X$,I,J)	MID$ called with three arguments returns a string expression composed of the characters of the string expression X$ starting at the Ith character for J characters. If I LEN(X$), MID$ returns a null string. If I or J<=Ø or> 255, an error occurs. If J specifies more characters than are left in the string, all characters from the Ith on are returned.
RIGHT$(X$,I)		Gives the rightmost I characters of the string expression X$. When I<=0 or> 255 an error will occur. If I =LEN(X=) then RIGHT$ returns all of X$
STR$(X)	290 PRINT STR$(X)	Gives a string which is the character representation of the numeric expression X. For instance, STR$(3.1)="3.1
CAL(X$)	280 PRINT VAL(X$)	Returns the string expression X$ converted to a number. For instance, VAL("3.1")=3.1. If the first non-space character of the string is not a plus (+) or minus (-)sign, a digit or a decimal point (.) then zero will be returned.

SPECIAL CHARACTERS

CHARACTER	USE	
π	120 PRINT π 140 A=π*2	Gives 3.14159265

RETURN	Return must end every line typed in. Returns print head or CRT cursor to the first position (leftmost) on line. A line feed is always executed after a carriage return.
STOP	Interrupts execution of a program or a list command. Stop has effect when a statement finishes execution, or in the case of interrupting a LIST command, when a complete line has finished printing. In both cases a return is made to BASIC's command level and READY is typed. Prints "BREAK IN LINE XXXX", where XXXX is the line number of the next statement to be executed.
: (colon)	A colon is used to separate statements on a line. colons may be used in direct and indirect statements. The only limit on the number of statements per line is the line length. It is not possible to GOTO or GOSUB to the middle of a line.
?	Question marks are equivalent to PRINT. For instance, ? 2+2 is equivalent to PRINT 2+2. Question marks can also be used in indirect statements. 10 ? X, when listed will be typed as 10 PRINT X. Do not use ?' with X to form PRINT X in other words always type out PRINT X in full, do not use ?' .
%	1ØA%=INT(X) Integer identifier. Designates an integer veriable in the range - 32767 to 32767

A P P E N D I C E S

****

PET CASSETTE FILE

First of all, find some suitable blank tapes.* At least three tapes are needed, and eight of them will let you get through the bulletin with a minimum of re-running or re-entering your programs.

Secondly, follow the directions EXACTLY. Do not take any 'short cuts', as these will lead you to some of the errors shown in PART III.

Third, Part IV describes the cassette related BASIC statements and variables in detail.

The BASIC statements for cassette files are:

OPEN	Open a file
CLOSE	Close a file
PRINT#	Write to a file
INPUT #	Read to a file
GET#	Read a single character from a file

The BASIC variable used for file status is:

ST	Status word

* Don't use the "three-for-a-pound" type tapes! We use a good, low noise, high energy tape.

CASSETTE (continued)

I. Some examples:

Example 1: Writing and reading numbers

Try out this program, being sure to type it in exactly as it appears here:

```
10 OPEN 1,1,1
20 FOR J = 1 TO 20
30 PRINT#1,J                Spell out the word PRINT .....
40 NEXT J                   Do not use ?#
50 CLOSE 1
60 PRINT "REWIND YOUR TAPE AND THEN PRESS A KEY"
70 GET A$: IF A$ = "" THEN 70
80 OPEN 1
90 FOR J = 1 TO 20
100 INPUT#1,X
110 PRINT X
120 NEXT J
130 CLOSE 1
140 PRINT "DONE"
```

Now list the program and compare each line with the listing above.

And finally, save the program on a cassette using the SAVE command:

SAVE "PGM 1"

Mark the cassette with the label "PGM 1". If you don't save and mark your program, you'll have to type it in again later. This Bulletin assumes from now on that you know how to save and load programs by name.

Remove the program cassette and put a fresh cassette in the recorder unit. Be sure the tape has been rewound, and then run the program. The screen will show

```
RUN

PRESS PLAY AND RECORD ON TAPE #1
```

When you have pressed the right buttons on the cassette unit, the screen will display "OK"

```
RUN
PRESS PLAY & RECORD ON TAPE *1
OK
```

The program will write data onto the tape, and when it is finished, you should see on your screen:

```
RUN
PRESS PLAY AND RECORD ON TAPE *1
OK
REWIND YOUR TAPE AND THEN PRESS A KEY
```

So ... rewind your tape and then press a key. Be sure the tape is fully rewound. Then, as the screen instructs, press PLAY on the cassette unit. The program now reads the numbers from the cassette and puts them onto the screen:

```
OK
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
DONE
READY
■
```

Cursor

Rewind the tape, label it "DATA 1", and put it away for the time being.

Now LIST the program. If you are experienced in BASIC, have patience, for here comes a line-by-line explanation of what the program does.

```
 30 PRINT#1,"I AM A PET!"      Again, don't use ?#
100 INPUT#1,X$
110 PRINT X$
```

Now RUN your new program and watch while the screen shows:

```
OK

I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
I AM A PET!
DONE

READY
```

LIST the program and check that lines 30, 100 and 110 are changed. Save this program as "PGM 2" on a fresh cassette. Don't destroy PGM 1 on the first cassette.

If you find this too dull, look at the descriptions of the statements at the back of this bulletin.

Line 10	Opens logical file 1 on tape drive 1, for write only (I/O option 1)
Lines 20 to 40	Write the integers 1 through 20 on the tape
Line 50	Closes the file

Lines 60 to 70	Tells you to rewind the tape and waits for you to press a key
Line 80	Opens logical file 1, with defaults of tape drive 1 for read only (equivalent to 80 OPEN 1,1,0)
Lines 90 to 120	Reads the numbers from tape and shows them on the display screen
Line 130	Closes the file
Line 140	Lets you know the program is completed correctly

Example 2: Writing and reading strings

Change the following lines in PGM 1. Hopefully, you still have the program in your PET. If you don't, load it from tape or type it again. In any event, when you're ready, type in these new lines, and PET will replace the old lines with the new ones.

Use a fresh cassette for the data, and label it "DATA 2".

From now on, we'll show the results, and not show the step-by-step displays, or mount/discount cassettes, etc. You should have four cassettes now: "PGM 1", "DATA 2", "PGM 2" and "DATA 2".

Example 3: Mixing strings and numbers

Make these changes to "PGM 2" and save the new program as "PGM 3".

```
 35 PRINT*1,J

105 INPUT *1,X

115 PRINT X
```

Note that lines 30, 100 and 100 are still present.

```
10
I AM A PET!
11
I AM A PET!
12
I AM A PET!
13
I AM A PET!
14
I AM A PET!
15
I AM A PET!
16
I AM A PET!
17
I AM A PET!
18
I AM A PET!   (cont'd)
```

```
19
I AM A PET!
20
DONE

READY
```

Run the program and save the data on a cassette marked "DATA 3". Now you have three programs and three data tapes. In the following sections, we will use these programs to show you the use of GET# and ST. Later on, some common errors will be examined.

II. Looking at data on tapes

First of all, read the descriptions of GET# and ST at the end of this bulletin. Then clear the program out of your PET by typing "NEW" and enter the following program EXACTLY:

```
 10 REM SHOW CONTENTS OF CASSETTE TAPES
 20 REM TO SOLVE TAPE MYSTERIES
 30 REM BY COMMODORE
 40 PRINT " ♥ -- SHOW TAPE PGM --
 50 PRINT
 60 PRINT "PUT YOUR DATA TAPE IN
 70 PRINT "CASSETTE #1 AND REWIND IT.
 80 GOSUB 1000
 90 PRINT "THE TAPE WILL BE READ AND
100 PRINT "SHOWN TO YOU IN 80 CHARACTER
110 PRINT "HUNKS.  WHEN YOU WANT TO STOP
120 PRINT "PRESS ANY KEY.  THE PROGRAM
130 PRINT "WILL ASK IF YOU WANT MORE
140 PRINT "DATA TO BE SHOWN.
150 GOSUB 1000
160 OPEN 1
170 PRINT " ♥ "      :H=0
180 H=H+1:PRINT"HUNK #"H
190 FOR J = 1 TO 80
200 GET#1,B$
210 IF ST > 0 THEN 400
215 IF ASC(B$) = 13 THEN PRINT "<RETURN>";:GOTO230
220 PRINT B$;
230 NEXT J
240 PRINT
```

The reverse field heart indicates a "clear Screen" character

```
250 GET A$
260 IF A$ = "" THEN 280
270 PRINT "MORE ?";
280 GET A$
290 IF A$ = ""THEN 280
300 IF A$ = "Y" THEN PRINT:GOTO180
310 END
400 PRINT:PRINT "STATUS WORD IS: ST"
410 IF (ST) AND 4 THEN PRINT "SHORT BLOCK
420 IF (ST) AND 8 THEN PRINT "LONG BLOCK
430 IF (ST) AND 16 THEN PRINT "READ ERROR
440 IF (ST) AND 32 THEN PRINT "CHECKSUM ERROR
450 IF (ST) AND 64 THEN PRINT "END OF FILE
460 IF (ST) AND 128 THEN PRINT "END OF TAPE
470 END
1000 PRINT:PRINT "PRESS ANY KEY
1010 GET A$: IF A$ = ""THEN 1010
1020 PRINT:RETURN
```

Note that the [■] in lines 40 and 170 is the "clear screen and home the cursor" character.

Save this program on a fresh cassette and label it "SHOW TAPE". You will find it handy for seeing what is on your tapes ... often what you <u>intended</u> to do is <u>not</u> what you did!

Now run this program, using the "DATA 1" tape. The CRT will show:

```
HUNK #1
1 <RETURN>   2   <RETURN>   3   <RETURN>   4   <RETURN>
5 <RETURN>   6   <RETURN>   7   <RETURN>   8   <RETURN>
9 <RETURN>  10   <RETURN>  11   <RETURN>  12   <RETURN>
13 <RETURN>  14   <RETURN>  15   <RETURN>  16   <RETUR
N> 17  <RETURN>  18   <RETURN>  19   <RETURN>  20   <R
ETURN>

STATUS WORD IS:  64
END OF FILE

READY
```

And here's the same program using the "DATA 2" tape:

```
HUNK #1
I AM A PET!   <RETURN>   I AM A PET!   <RETURN>
I AM A PET!   <RETURN>   I AM A PET!   <RETURN>
I AM A PET!   <RETURN>   I AM A PET!   <RETURN>
I
HUNK #2
AM A PET!   <RETURN>   I AM A PET!   <RETURN>   I
AM A PET!   <RETURN>   I AM A PET!   <RETURN>   I
AM A PET!   <RETURN>   I AM A PET!   <RETURN>   I
AM
MORE? ■
```

Here, a key was pressed during HUNK #2. Some of the critical lines in "SHOW PROGRAM" are:

20 GET #1,B$	This gets the character from the file
210 IF ST>0 THEN 400	If any file condition is encountered, this jumps to a report of the condition. Since GET# will read past end-of-file marks, the status must be checked each time a character is read.
215	Detects RETURN and displays it in a visible form.
400 - 460	Reports status. Note the "AND" is a logical mask operation which checks for the appropriate bit in ST.

Run this program with "DATA 3" and see if the file looks like you expect it to look.

III. Some Examples that Don't Work

It is easy to make errors with cassette files. Some will give a ?SYNTAX ERROR and others will stop BASIC and force you to turn the PET's power off and start again. Be sure you have the tapes "SHOW TAPE", "PGM 1" and "DATA 1" available before you start this section.

ERROR #1: THE "?#" SHORTCUT

Load "PGM 1" and change line 30. Type in: 30 ?#1,J

Now try to run it ...

?SYNTAX ERROR IN 30

So LIST 30

30 PRINT#1,J

Mysterious, isn't it? Now you know that "?#1" does not work. You must always spell out the word PRINT# when using cassette files.

Okay, here's the explanation. When you typed in line 30, BASIC converted the PRINT, or "?", into a token. However, the tokens for PRINT and PRINT# are different.

Suppose PRINT becomes [X]

and PRINT# becomes [Y]

Then,if you type in 30 PRINT J BASIC stores it as 30 [X] J, and if you type in 30 PRINT#1,J BASIC stores it as 30 [Y] 1,J. 30 ?J becomes 30 [X] J --- but 30 ?#1,J becomes 30 [X] #1,J. So, 30 ?#1,J LISTS as 30 PRINT#1,J and looks correct. However, when it is run, BASIC sees the # following the PRINT token. Since # is not a number or a legal variable name, BASIC gets upset and tells you you have a syntax error. In line 30.

REMEMBER!!!!! IF YOU GET A SYNTAX ERROR IN A PRINT TO A FILE STATEMENT, AND IT LOOKS OK WHEN YOU LIST IT, THE FIX IS:

RETYPE THE LINE USING
THE FULLY SPELLED WORD

<u>PRINT#1</u>

IV. The Output Image

The PRINT # statement prints exactly as it is told to. If more than 40 characters are output to a file without a carriage return, no carriage return will be inserted. Type in the following program:

```
10 OPEN 1,1,1
20 X$="1234567890"
30 FOR J + 1 TO 5
40 PRINT X$;
50 PRINT 1,X$;
60 NEXT J
70 CLOSE 1
```

Put the "DATA 1" tape in the cassette drive, rewind it, and run the program. Then load "SHOW TAPE" and use it to look at the "DATA 1" tape. Notice that when you run the first program, the screen looked like this:

```
PRESS PLAY AND RECORD ON TAPE  1
OK
12345678901234567890123456789012345678901
1234567890

READY
```

No carriage return was put here!

The line ran off the right edge of the screen and appeared on the next line, but no carriage return was ever printed. The "SHOW TAPE" program proves this: No <RETURN> appears in HUNK #1.

Try a few more combinations. PRINT# will always write what it is told to write on the tape. Remember that PRINT# writes on the tape just like PRINT does on the screen (if you had a mile wide screen, that is).

V. The Input Image

The INPUT statement in PET BASIC has some oddities. To understand this, some examples without using cassettes are in order.

<u>Example 1. Discard of Extra Input</u>

```
10 INPUT A,B,C
20 PRINT A,B,C
```

RUN, and enter 1,2,3,4,5 when the question mark appears on the screen. The screen will show:

```
RUN
? 1,2,3,4,5
? EXTRA IGNORED
  1           2           3

READY
```

<u>Example 2. 80 Character INPUT limit</u>

Try:

```
10 INPUT X$
20 PRINT X$
```

Run, and enter AAAAAAAAAA.....until you have 100 "A"s entered.

The screen will show:

```
RUN
? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAA

READY
```

PET printed this line

This is because the input buffer can only accept 80 characters at a time. If more are entered, the first 80 are lost.

<u>Example 3: Too Little INPUT</u>

Try:

```
10 INPUT A,B,C,D,E,F
20 PRINT A;B;C;D;E;F
```

Now RUN, and enter 1,2,3 RETURN
4,5,6 RETURN and you get:

```
RUN
? 1,2,3
?? 4,5,6
 1  2  3  4  5  6

READY
```

INPUT will look past a carriage return until all the variable list is satisfied (A,B,C,D,E,F in the example). Now try this one again, but enter 1 2 3 4 5 6 RETURN

You will get a ??. Enter RETURN until you see READY. INPUT ignores blanks when reading numbers and will keep asking for more until it is finished. Note that a number for A is 123456.

This digression really will help you understand the following rules for writing cassette file data.

RULES FOR WRITING ON CASSETTE FILES

1. Be sure to have matching INPUT# and PRINT# variable lists. If your INPUT list is too short, you will lose data.

2. Don't ever print more than 79 characters without a carriage return. If INPUT# reads over 80 characters, it will either lose the first 80 characters or CRASH BASIC!

3. Extra carriage returns don't hurt anything.

4. If you want to write several numbers on a line, separate them with a comma "," or else, when they are read, you will get the wrong values.

Here are some examples to show what happens with tapes. In each example, load your "PGM 1" and modify it accordingly. Don't save it, though, since each example assumes you use the original "PGM 1". Then use "SHOW TAPE" to see what is on your data tape afterwards. Use the "DATA 1" tape for these examples.

ERROR#2 No carriage returns or commas between numbers

Type 30 PRINT#1,J; and run it. The program will eventually show:

```
OK
1.23456789E+31
?OUT OF MEMORY ERROR IN 8224
FORMULA TOO COMPLEX ERROR IN 8224
```

and CRASH! You have to turn the power off and start again. This is sometimes unpredictable. With "DATA 1" you will see 1 2 3 4 5 6 . . . . . without carriage returns or commas.

ERROR#3 Loss of additional INPUTs

Load "PGM 1" and type 30 PRINT#1,J","; and run it .....

a "1" appears

and CRASH! Again, INPUT# tried to read past the end of the file, looking for a <RETURN> , with disastrous results.

Change line 105: 105 IF ST>0 THEN PRINT "END OF FILE":GOTO140

You will get: END OF FILE
DONE

So ..... If you test the status after INPUT# you can avoid a crash, though more than likely you will not have done what you wanted to.

ERROR#4 More than 80 characters in a line

Again, load "PGM 1" and change it as follows:

```
 30 PRINT#1 "ABC123";
 90 |
120 | Delete these lines
100 INPUT#1,X$
110 PRINT X$
```

Now run it . . . .
and CRASH! (Yet again. I hope you can see why the rules are to be followed!)

ERROR#5 Substring File name matching

If you open a file with a name, the PET will read the tape until it finds a suitable name. Enter the following program:

```
 10 OPEN 1,1,1,"FILE1"
 20 PRINT#1, "THIS IS FILE1"
 30 CLOSE 1
 40 OPEN 1,1,1,"FILE"
 50 PRINT#1,"THIS IS FILE FILE"
 60 CLOSE 1
 70 PRINT "FILENAME";
 80 INPUT F$
 90 PRINT "REWIND YOUR TAPE . . . ."
100 GET A$: IF A$ = "" THEN 100
110  OPEN 1,1,0,F$
120 INPUT#1, X$
130 PRINT X$
140 CLOSE 1
```

Notice that you have to include all the parameters here

If you get a syntax error in 10 (or 20 or 110) chances are you forgot the last comma!

Run this program and enter "FILE" as the filename (F$). You will see the contents of FILE1 displayed.

So . . . Be sure you file names do not match each other even in substrings like: COM and COMMODORE, or MODE and REMODEL.

ERROR#6 Reading a program as data

Enter this program. When you run it, use your "PGM 1" tape for the data tape.

```
 10 OPEN 1
 20 INPUT#1,X$
 30 PRINT X$
 40 CLOSE 1
```

You will get a BREAK in 10 message. If you type ?ST, you will see a 0 because the file never opened successfully.

ERROR#7 Going past End-of-File with GET#

Load "PGM 1" and run it using "DATA 1". Then do a "NEW" and enter this program:

```
10 OPEN 1
20 GET#1,X$
30 PRINT X$;
40 GOTO 20
50 CLOSE 1
```

The numbers 1 to 20 will appear, then a 1, then the cassette goes on. It will go on with garbage or other stuff appearing; you have done a lot with "DATA 1"! Now, you can fix it with:

```
25 IF ST>0 then 50
```

and run it again. Note: You may stop functioning (well, your PET may) and have to start over again. Halting during a tape read is hazardous!

NOW it works!

ERROR#8 Not fully rewinding the tape

In many instances, people fail to fully rewind a tape, and the program fails to read it. So be careful. If you fail after checking your program carefully and after three careful attempts to load, you may have a hardware problem.

CASSETTE RELATED BASIC STATEMENTS AND VARIABLES

OPEN Opens a file for input/output. The syntax is:

OPEN (logical file*), (physical device number), (I/O option), (filename)

The keyword "OPEN" and the logical file number are required. The other items are optional. If they are not specified, a default value will be used.

Logical File* This number is used in the CLOSE, PRINT, INPUT and GET statements to refer to this file. The logical file number can be from 1 to 255. Up to 10 files may be open at the same time.

NOTES: If you try OPEN,0 you will get a syntax error
OPEN,-1 or OPEN,256 gets "ILLEGAL QUANTITY ERROR"

If you get more than 10 files at once, the PET will "hang" and you will have to turn off the power. If you open a file with the same logical file number as one which is already open, you will get a "FILE OPEN ERROR".

Physical Device Number For cassettes, the numbers 1 and 2 are legal. Number 1 refers to the cassette in the PET and 2 is for the auxiliary, or external cassette unit. The default value is 1.

NOTES: The physical device number may be from 0 to 255. The PET currently recognizes devices 0 - 15. If a device number is out of range, you will get an "ILLEGAL QUANTITY ERROR".

I/O Option This tells the cassette whether to read or write to the file. 0 is for read only, 1 is for write only, and 2 is for write only with end-of-tape marker. The default value is 0 (read).

NOTES: If option 2 is used, an EOT will be written on the tape when the file is closed. An error will result if, at a later time, you attempt to read past the EOT mark.

WARNING:	WARNING:	WARNING:	WARNING:	WARNING:

A. If you use physical device numbers other than 1 or 2, you must follow the rules for that device. For instance, the IEEE buss conventions are different from the cassette rules.

B. If you attempt to use

```
OPEN
OPEN 1,
OPEN,1
OPEN,,1
```

etc., you will get a syntax error.

FILENAME The filename is used to identify a file. It may be up to 187 characters long. However, please note that:

1. If you ask in your program for a filename, the longest string you can input is 80 characters.

2. If you are searching for a file with a short name, and your tape has a file with a long name on it, the search looks for a matching sub-string. This means that if you are looking for a file named "CAT", and the tape has on it a file named "CONCATENATE", it will be recognized as "CAT".

3. Only the first 16 characters of a filename will be displayed after the "SEARCHING" message appears on the screen.

To summarize,

* Keep your filenames short (preferably under 16 characters)

* Avoid files named alike, such as

ACCOUNT ACCOUNTS ACCOUNTING	If you are looking for a file named "COUNT", any of these files will be recognized as correct files.

* The default filename is a null string "". An "OPEN 1" will open any file, as null is always recognized as a filename, regardless of the name of the file.

CLOSE This tells the PET to:

1. Stop reading a file	or,	Open Option 0
2. Stop writing a file and make an end-of-file mark	or,	Open Option 1
3. Stop writing a file and make an end-of-file mark and make an end-of-tape mark		Open Option 2

depending on how the file was opened.

All files which have been opened should be closed before a program ends and before you remove the cassette. If you do not close the file, your files may become garbled. For example, if you are writing a file on a previously used tape, and you don't close it, then when you read it later, you will either have a read error or you will read past the files' end into the old (and meaningless) junk on the tape.

PRINT# (logical file#), (variable list)

PRINT* writes an exact copy of the characters produced by an equivalent PRINT statement. This includes graphics, upper/lower case and cursor control characters.

NOTES: 10 FOR J = 1 TO 100:PRINT "X";:NEXT J will print 100 successive "X"s on the display. Though this will appear as 2-1/2 lines of "X" on the screen, no carriage returns are present at the ends of the first two lines. When 10 FOR J = 1 TO 100: PRINT*1,"X";:NEXT J is executed, 100 successive "X"s are written onto the cassette with no carriage returns.

Some warnings are worth noting. First,*? will not work. If you get a syntax error and the LIST gives a correct appearing line, try retyping the line using PRINT*. Do not use the screen editor unless you type "PRINT" over the PRINT which appears on the screen.

If you intend to read the file later,using the INPUT* command, be sure that the carriage returns are liberally included and that no more than 79 characters in succession appear without a carriage return. This applies only to INPUT*, and not to GET*.

INPUT* (logical file*), (variable list)

This reads tape exactly as if it were the keyboard. As with keyboard input, a maximum of 80 characters, including the carriage return, may be entered.

This means the tape cannot have more than 79 successive characters without a carriage return, if it is to be read successfully with INPUT*.

WARNING: WARNING: WARNING: WARNING: WARNING:

If you attempt to INPUT* from a tape with more than 79 characters between carriage returns, BASIC will either go away entirely (CRASH) and you'll have to turn the power off and start again, or strange errors and unidentified flying glitches may appear.

Note that 79 character limitation is due to an 80 character input buffer and is not currently modifiable.

GET* (logical file*), (string or numeric variable)

This reads the tape one character at a time. There are two varieties:

1. GET* (logical file*), (numeric variable)

- if the character in the file is a digit (0 - 9), then the numeric variable will be set to the value of the digit.
- if the character is one of these: + - blank then the numeric variable is set to zero.

- any other character or the end-of-file marker will produce a syntax error. And THIS error will NOT tell you the line number in which it occurs!

2. GET# (logical file#), (string variable)

This reads the file, one character at a time, and returns the character in the string variable as a one character string. If the status word is not checked, successive applications of GET# will read the file past the end-of-file mark.

NOTE: It is not necessary to write carriage returns at 79 characters or less if you read your data back with a GET# command.

STATUS WORD

The Status word can be checked after each I/O operation for certain conditions.

To detect the status, use the "AND" operation in BASIC as shown in the example in this bulletin. Another way to do it is 10 GET#1, A$:B$=B$+A$:IF NOT (ST)AND64)THEN 10 which will read data into B$ until an EOF is encountered.

The status word ST is updated each time there is an I/O operation, with a code indicating the outcome of that operation. To indicate a unique condition, one bit is set at a time. Multiple bits may be set so it is necessary to break down the decimal number into its binary powers to determine which bits were set. For example, if ST = 56, then bits 8, 16 and 32 were set: 56=32+16+8

STATUS CODES FOR TAPE I/O

4	Short Block
8	Long Block
16	Unrecoverable read error
32	Checksum error
64	End of file
128	End of tape

SHORT BLOCK (4) When reading a block from tape, shorts (the delimiter between blocks) were encountered before the expected number of bytes had been read from that block. Possible cause: attempting to read a short load file as a data record.

LONG BLOCK (8) When reading a block from tape, shorts were not encountered after the expected number of bytes had been read from that block. Possible cause: reading a long load file as data.

UNRECOVERABLE READ ERROR (16) **FATAL ERROR** Return to BASIC and print error message.. Cause: More than 31 errors on the first block of redundant blocks - or - an error that could not be corrected because it occured in the same place in both blocks.

CHECKSUM ERROR (32) After a LOAD or reading of data, a checksum is computed over the bytes in RAM and compared to a byte received from the input device. If they do not match, this bit is set. Possible Cause: faulty RAM - or - multiple bit error in data transmitted. This bit is also set if data or program fails a verify operation.

END OF FILE (64) This bit is set when an attempt to read data from a tape file is made when there is no more data.

END OF TAPE (128) **FATAL ERROR** An EOT record was found before the file being searched for was encountered.

SPACE HINTS

In order to make your program smaller and save space, the following hints may be useful.

1) Use multiple statements per line. There is a small amount of overhead (5bytes) associated with each line in the program. Two of these five bytes contain the line number of the line in binary. This means that no matter how many digits you have in your line number (minimum line number is Ø, maximum is 64000), it is still two bytes. Putting as many statements as possible on a line will cut down on the number of bytes used by your program.

2) Delete all unnecessary spaces from your program. For instance:

```
10 PRINT X, Y, Z
```

uses three more bytes than

```
10 PRINTX,Y, Z
```

Note: All spaces between the line number and the first non-blank character are ignored.

3) Delete all REM statements. Each REM statement uses at least one byte plus the number of bytes in the comment text. For instance, the statement 130 REM THIS IS A COMMENT uses up 24 bytes of memory.

In the statement 140 X=X+Y: REM UPDATE SUM, the REM uses 14 bytes of memory including the colon before the REM.

4) Use variables instead of constants. Suppose you use the constant 3.14159 ten times in your program. If you insert a statement

```
10 P=3.14159
```

in the program, and use P instead of 3.14159 each time it is needed, you will save 40 bytes. This will also result in a speed imporvement.

5) A program need not end with an END; so, an END statement at the end of a program may be deleted.

6) Reuse the same variables. If you have a variable T which is used to hold a temporary result in one part of the program and you need a temporary variable later in your program, use it again. Or, if you are asking the user to give a YES or NO answer to two different questions at two different times during the execution of the program, use the same temporary variable A$ to store the reply.

7) Use GOSUB's to execute sections of program statements that perform identical actions.

8) Use the zero elements of matrices; e.g. A(0), B(0,X).

STORAGE ALLOCATION INFORMATION

BASIC	BYTES	USED FOR
BASIC	1028	1/O buffers Tables Scratch Pad
Line Numbers	4	
BASIC keywords	1	
Characters	1 ...	Includes RETURN
Variables	7 ...	If a value is assigned (for integers and floating point variables)
	7 +	length of string for string variables
Arrays	* (S+1) + (2*D)	
(Size includes the 0th element)	Where:	
	A = 5	Floating point array
	A = 3	String array
	A = 2	Integer array
	and S = ...	Size of array
	D = ...	Number of dimensions

When a program is being executed, space is dynamically allocated on the stack as follows:

a) Each active FOR ... NEXT loop uses 22 bytes

b) Each active GOSUB (one that has not returned yet) uses 6 bytes

c) Each parenthesis encountered in an expression uses 4 bytes and each temporary result calculated in an expression uses 12 bytes.

9) Use integer variables or arrays - A%, HX% (I,J)etc. wherever possible.

SPEED HINTS

The hints should improve the execution time of your BASIC program. Note that some of these hints are the same as those used to decrease the space used by your programs. This means that in many cases you can increase the efficiency of both the speed and size of your programs at the same time.

1) Delete all unneccesary spaces and REM's from the program. This may cause a small decrease in execution time because BASIC would otherwise have to ignore or skip over spaces and REM statements.

2) THIS IS PROBABLY THE MOST IMPORTANT SPEED HINT BY A FACTOR OF 10. Use variables instead of constants. It takes more time to convert a constant to its floating point representation than it does to fetch the value of a simple or matrix variable. This is especially important with FOR ... NEXT loops or other code that is executed repeatedly.

3) Variables which are encountered first during the execution of a BASIC program are allocated at the start of the variable table. This means that a statement such as 5 A=0: B=A: C=A, will place A first, B second and C third in the symbol table (assuming line 5 is the first statement executed in the program). Later in the program, when BASIC finds a reference to the variable A, it will search only one entry in the symbol table to find A, two entries to find B and three entries to find C, etc.

4) NEXT statements without the index variable, NEXT is somewhat faster than NEXT 1 because no check is made to see if the variable specified in the NEXT is the same as the variable in the most recent FOR statement.

DERIVED FUNCTIONS

The following functions, while not intrinsic to PET BASIC, can be calculated using the existing BASIC functions:

FUNCTION	FUNCTION EXPRESSED IN TERMS OF BASIC FUNCTIONS
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN (X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X) = -ATN(X/SQR(-X*X+1))+1.5708
INVERSE SECANT	ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COTANGENT	ARCCOT(X) = -ATN(X)+1.5708
HYPERBOLIC SINE	SINH(X) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1
INVERSE HYPERBOLIC SINE	ARGSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARGCOSH(X) = LOG(X+SQR(X*X-1))
INVERSE HYPERBOLIC TANGENT	ARGTANH(X) = LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARGSECH(X) = LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARGCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)
INVERSE HYPERBOLIC COTANGENT	ARGCOTH(X) = LOG((X+1)/(X-1))/2

CONVERTING BASIC PROGRAMS NOT WRITTEN FOR THE PET

Though implementations of BASIC on different computers are in many ways similar, there are some incompatibilities which you should watch for if you are planning to convert some BASIC programs that were not written for the PET.

1) Matrix subscripts. Some BASICs use " ("and") " to denote matrix subscripts. PET BASIC uses " ("and") ".

2) Strings. A number of BASICs force you to dimension (declare) the length of strings before you use them. You should remove all dimension statements of this type from the program. In some of these BASICs, a declaration of the form DIM A$(I,J) declares a string matrix of J elements each of which has a length 1. Convert DIM statements of this type to equivalent ones in PET BASIC: DIM A$(J),

PET BASIC uses " + " for string concatenation, not " , " or " & ".

PET BASIC uses LEFT$, RIGHT$ AND MID$ to take substrings of strings. Other BASICs use A$(I) to access the Ith character of the string A$, and A$(I,J) to take a substring of A$ from character position I to character position J. Convert as follows:

OLD	NEW
A$(I)	MID$(A$,I,1)
A$(I,J)	MID$(A$,I,J-I+1)

This assumes that the reference to a substring of A$ is in an expression or is on the right side of an assignment. If the reference to A$ is on the left hand side of an assignment, and X$ is the string expression used to replace characters in A$, convert as follows:

OLD	NEW
A$(I)=X$	A$=LEFT$(A$,I,1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I+1)+X$+MID$(A$,J+1)

3) Multiple assignments. Some BASICs allow statements of the form: 500 LET B=C=0. This statement would set the variables B & C to zero.

In PET BASIC, this has an entirely different effect. All the " = 's " to the right of the first one would be interpreted as logical comparison operators. This would set the variable B to -1 if C equaled 0. If C did not equal 0, B would be set to 0. The easiest way to convert statements like this one is to rewrite them as follows:

```
500 C=0: B=C
```

4) Some BASICs use " / " instead of " : " to delimit multiple statements per line. Change the " / "'s to " : "'s in the program.

5) Programs which use the MAT functions available in some BASICs will have to be re-written using FOR ... NEXT loops to perform the appropriate operations.

ASCII CHARACTER CODES

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	↑
009	HT	052	4	095	←
010	LF	053	5	096	`
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

LF=Line Feed FF=Form Feed CR=Carriage Return DEL=Rubout

COMMODORE

Commodore Systems Division
446 Bath Road, Slough, Berkshire, SL1 6BB
Telephone Burnham (06286) 3224/5/6
Telex 848316

BASIC BUGS

We'll publish all the bugs we know about, and a few months from now, when we've found and fixed all of them, we'll produce a new ROM which you'll be able to buy and plug in.

1. 10 IF F OR 1 = 10 THEN 10 gets collapsed to

   10 IF FOR I = 10 THEN 10 and yields a ?SYNTAX ERROR

We've found and fixed this one. The only reserved word that can have embedded spaces is GOTO, which may appear as GO TO. Therefore, "FOR" in this example will no longer be converted to a reserved word.

2. The BYTES FREE message number and the amount of bytes free when PRINT FRE(0) is typed just after start-up are different.

This is not a bug. PRINT FRE(0) uses 3 bytes of RAM.

3. The SAVE command should respond with "PRESS REC AND PLAY" instead of "PRESS PLAY AND REC", since the latter sequence doesn't work.

This is liveable and probably won't be fixed.

4. The POS function is not effected by Pokes and other cursor movements. It does not keep track of where the cursor is moved with POKEs and other cursor movements.

POS will be deleted from BASIC.

5. SPACE and shifted SPACE characters have different ASCII values. This is not a bug. Shifted and unshifted characters are indiced separately.

6. When a quotation mark (Code 34 or 98) is output, the rest of the line treats cursor movement literally. Example:

```
10 PRINT CHR$(34), 34      (Try it and see)
```

This will not be changed at present.

7. SPC(0) returns 256 spaces.

Fixed.

8. Direct lines beginning with colons,":", are ignored.

Fixed.

9. Arrays with more than 255 elements fail.

Fixed.

10. Random Number Function -- How does it work?

NAME	EXAMPLE
RND (X)	170 PRINT RND (X)

Generates a random number between 0 and 1. The argument X controls the generation of random numbers as follows:

X > 0 generates a new sequence of random numbers using X as a seed. Calling RND with the same X where X 0 will generate the same random for each X if X does not change.

Example: RND (-1) gives
2.99196472E-08 for as many times as you use -1.
2.99205567E-08 for as many times as you use -2.

This is useful for debugging where you want the same random number to be generated. You can get a different but constant random number with any minus number.

X = 0 generates .564705882 each time you call

X > 0 will generate the next randomly sequenced random number if X does not change. If X changes, the new X is used as a seed to a new sequence of random numbers.

If you want to verify what the RND actually does, enter the program:

```
10 INPUT     R
20 X=RND    (R)
30 PRINT     X
40 GOTO     10
```

Then try various values for the input.

11. CHR$ accepts string arguments.

Fixed.

12. DEF FN fails in one out of 256 cases.

Fixed.

The following is an example of a PET QUIRK. It is not a bug, and happens because the character printed after a number is a "cursor right" rather than a carriage return or a space.

Output of a number is: | SIGN | N | U | M | B | E | R | CURSOR RIGHT |

Which can cause havoc with screen overwrites if you aren't aware of it.

```
10 PRINT "S";
20 FOR I = 1 to 10
30 PRINT "BBBBBBBBB"
40 NEXT
50 PRINT "S"
60 FOR I = 1 to 10
70 PRINT I*100*HI!"
80 NEXT
```

The black "S" on a white field is the character used to represent "home cursor".

And lo! on your screen will appear:

```
100BHI! BBB
200BHI! BBB
. . . . . . . . .
```

SOME QUESTIONS AND ANSWERS

QUESTION: Will COMMODORE help me design a program or a system for my specific application?

ANSWER: No. Manuals and bulletins are in the works; a software library will be available soon. COMMODORE cannot afford to help each person design for his specific needs and offer such an incredible price.

QUESTION: How do I get an array of graphics to print on the CRT?

ANSWER: Use a semicolon in your PRINT statement (PRINT "...";). It will look like you built a string and printed it. Maximum string length is 255 characters, and prints in 3.2 lines. If you're trying to print all of PET's graphics, you may be blanking the screen when you try to print CHR$(146), which is a "clear screen" character.

QUESTION: PET prints .003 as 3E-03. Can I suppress scientific notation?

ANSWER: Yes. You'll have to write a formatting program to do it.

QUESTION: Why are the squares of integers not integers? For example, 7 ↑ 2=49.0000001 while EXP(LOG(7)*2)=49.

ANSWER: Logarithms are use and there are built-in round off problems in binary representation of decimal numbers.

QUESTION: How can I get around the 255 element array limitations?

ANSWER:

A) For multidimensional arrays, use separate arrays. For example, DIM A(100,3)→ DIM A(100), B(100), C(100).

B) Pack your values, two or three to an element.

C) Change your algorithm to not require arrays.

QUESTION: Can PET do matrix arithmetic?

ANSWER: You will have to write a program to do it.

QUESTION: Why doesn't the OTHELLO program from October BYTE work?

ANSWER: To many GOSUBS without returns. PET can only accept 26 levels of GOSUB nesting.

QUESTION: Can I write my own tape header?

ANSWER: Yes. Just SAVE"FILENAME", then LOAD"FILENAME". Or, to use data files, OPEN 1, 1, 1,"FILENAME". FILENAME" can be a string; i.e., F$.

QUESTION: How can I create a data file on tape?

ANSWER: The Cassette Bulletin will give you the answer to this one.

QUESTION: Do you have a Fast Forward off a cassette leader before saving a program?

ANSWER: No. The operating system software provides about 7.5 seconds to move the tape off the leader before beginning recording of data.

QUESTION: How fast is cassette data storage?

ANSWER: The data rate is 30-50 CHAR/SEC.

QUESTION: What is the tape format?

ANSWER: The format is a unique COMMODORE scheme.

QUESTION: What does PET look for on tape when it searches?

ANSWER: The header block on the tape file.

QUESTION: How many files can be open at one time?

ANSWER: Ten. More than ten will hang the PET up and you will have to turn the power on and off.

QUESTION: Where are the cassette buffers?

ANSWER: Cassette # 1 from $027A to $0339) $ means this
Cassette # 2 from $033A to $03FE) is a hexadecimal number

QUESTION: How is End-of-Memory determined by BASIC?

ANSWER: On Power-up reset, a checkerboard patters is written and read back while incrementing a pointed until failure occurs. The highest memory location is pointer - 1.

QUESTION: How do you delete a line?

ANSWER: Type the line number only, then press RETURN.

QUESTION: Will trig functions work on arguments in degrees?

ANSWER: SIN, COS, and TAN require arguments in radians. Convert degrees to radians by multiplying:

degrees * π /180

Remember, π is a constant available from the keyboard.

QUESTION: What will happen if I try mixed mode arithmetic?

ANSWER: All arithmetic is performed in floating point. If an operation is performed on an integer, it is first converted to floating point, and if assigned to an integer variable, the result is appropriately truncated or left alone.

QUESTION: Can you program in machine language from BASIC and not use a monitor?

ANSWER: Yes. By using the POKE command, it is possible to load RAM. The process can be automated with a BASIC loader program which contains the bytes of the machine code program in DATA statements. To be safe, poke into casette buffer # 2.

QUESTION: How is SYS used?

ANSWER: The parameter for SYS is a decimal address. This is evalated and used as a target for a JMP instruction. Return to BASIC via RTS.

QUESTION: How is USR used?

ANSWER: 1. POKE the address of the subroutine

location 1 gets the low byte

location 2 gets the high byte

2. Call USR (i.e., A=USR(I))

3. The parameter is evaluated and placed in the floating accumulator.

4. The function value is returned in the floating accumulator.

5. Return to BASIC via RTS.

QUESTION: How do you get lower case letters?

ANSWER: POKE 59468,14 for lower case
POKE 59468,12 for graphics

Lower case letters and graphics cannot be displayed on the screen simultaneously. Only use masks 12 and 14 or you may disable the keyboard interrupts. The POKE command sets a chip address select on the character generator ROM.

QUESTION: Where is BASIC text in memory?

ANSWER: It begins at $0400 and extends to $0FFF or $1FFF, depending on whether it is a 4K or an 8K PET.

QUESTION: Where are variables stored, and can they be passed from one program to another?

ANSWER: During program execution, strings are created and stored downward from highest memory. Integers and real numbers are stored upward from the end of BASIC text. They may be passed to an overlay program if the overlay is less than or equal in size to the program which initiated the LOAD.

QUESTION: How do I use the diagnostic routines?

ANSWER: Special hardware is required which is currently available only to dealers and service people.

QUESTION: Where and when can I get the necessary hardware to run the diagnostic routines?

ANSWER: Only authorized PET service people will have the required hardware for the present.

QUESTION: Can I get an O.S. source listing or a BASIC source listing?

ANSWER: This will be discouraged for a purpose of maintaining software compatibility between PET users.

QUESTION: What level of BASIC is provided in PET's ROMs?

ANSWER: PET BASIC is very close to MITS BASIC by Microsoft, and has been expanded in the area of I/O and arithmetic precision.

QUESTION: Does PET have a SORT function?

ANSWER: No. SORTing must be done by a BASIC program. See Knuth, "The Art of Computer Programming" for a variety of algorithms.

QUESTION: Is PET base page limited?

ANSWER: No. At the BASIC programming level this is transparent to the user. In machine code programming page 0 is always at a premium.

QUESTION: How does PET compare strings?

ANSWER: In alphabetical order according to ASCII code, for example,

"A"<"AA" and "ABCD"< "ABCE"

QUESTION: Is the screen refreshed from a specific IK of memory?

ANSWER: Yes, starting at $8000.

QUESTION: Can I POKE the locations for cursor control?

ANSWER: We do not recommend using POKE to control the cursor. The cursor is controllable from the keyboard cursor control keys, and from Basic.

QUESTION: Can PET be reset withour destroying RAM content?

ANSWER: No.

QUESTION: What is the PET's power consumption?

ANSWER: Less than 100 watts.

QUESTION: Why is the PET only expandable to 32K RAM?

ANSWER: Because the upper 32K is reserved for O.S., I/O, and ROM, and the 6502 can only address 65K.

QUESTION: Is the 6502 a tristate chip?

ANSWER: The 6502 has some lines which are tristate and some which aren't. Contact MOS Technology for specs.

QUESTION: Does PET disrupt TV or radio?

ANSWER: PET is extremely well shielded and emits very little RF interference. The only time you may notice it is if you place a TV set inches away from PET and tune to an extremely weak station. Try that with a pocket calculator or a digital clock!

QUESTION: How do you access the user port?

ANSWER: The user port is on a MOS 6522. The simplest I/O is accomplished by POKEing a data direction register at 59459 and then PEEKing or POKEing a data register at 59471. The I/O logic levels are TTL.

QUESTION: Can the IEEE-488 be adapted to S-100?

ANSWER: The IEEE-488 is an I/O peripheral bus. The S-100 is a memory bus. They are not the same thing. PET does have a memory expansion bus which can be adapted to drive many S-100 periperals.

QUESTION: What changes need to be made to an HP printer to get it to work on the PET?

ANSWER: Most HP instruments work on the IEEE bus which PET supports. We have tested a thermal printer (HP 5150A) and an impact printer (HP 9871A) successfully. Use an edge-card connector instead of the standard pin connector.

QUESTION: How do you get a listing on an IEEE printer?

ANSWER: Essentially: Open the file, tell the device to "listen", and then LIST.

OPEN 4,4	establish output channel - open file
CMD 4	create alternative output device - tell the device to listen
LIST	list to that device
CLOSE 4	close alternate channel - close the file

QUESTION: Can I use someone else's RAMs to build my own memory board?

ANSWER: Yes. See the memory expansion pinout.

QUESTION: Can PET be hooked to a terminal?

ANSWER: An IEEE-488/RS232 interface is in the works.

QUESTION: What causes my PET's CRT to get the 'jitters'?

ANSWER: Probably the 12v regulator. Write to PET service.

QUESTION: Why won't my PET load and save my program?

ANSWER:

1. Are you using bad tapes?
2. Have you fully rewound the tape before a save or load?
3. Have you recently cleaned and demagnetized the deck heads?
4. If every one of these questions is answered correctly and PET still won't read tapes, it could be due to poor alignment to the read/record heads. Check with PET service.

QUESTION: If the RETURN key glitches out in the middle of a program, How can I save myself? Do I HAVE to reset?

ANSWER:

1. If the cursor can be seen, press RETURN.
2. If the cursor can't be seen, press the RUN/STOP key.
3. If neither works, you must reset. Check for possible hardware mulfunction. Is the keyboard connector firmly attached to the main board?
4. And, if all else fails, check to be sure you haven't left any tape or printer files open. PET may be sending the RETURN to the file.

QUESTION: Will COMMODORE be bringing out a bigger CRT?

ANSWER: Probably not. You can use your own monitor on the user port. See the pinout for this port in this issue.

QUESTION: Will COMMODORE be making a cassette with a counter?

ANSWER: Possibly, but not for a while.

QUESTION: Will COMMODORE be making a bigger keyboard?

ANSWER: Yes. On a bigger PET. With a bigger price tag.

QUESTION: If I buy a printer from someone else, will COMMODORE help me get it running on the PET?

ANSWER: No. An RS232/IEEE-488 interface is in the works.

QUESTION: Will COMMODORE provide a disassembler as part of the purchased package?

ANSWER: The disassembler is already in the public domain, and you can buy it.

QUESTION: What peripheral is COMMODORE planning for the future?

ANSWER: Second cassette, printer, floppy disk, telephone modem, and memory board, to start. More will be contemplated later.

commodore

COMMODORE BUSINESS MACHINES, INC.
901 CALIFORNIA AVENUE
PALO ALTO, CALIFORNIA 94304
TELEPHONE: (415) 326-4000 TELEX: 345-589
CABLE ADDRESS COMBUSMAC PLA

USR

The USR function allows a programmer to create a machine language subroutine which is callable from BASIC. USR has a parameter which is evaluated and placed in the floating accumulator at location $BØ. The format is as follows:

```
$BØ  -  exponent  +  $8Ø
                          normalized so
$B1  -  mantissa     MSB  B7 set
$B2  -    "
$B3  -    "
$B4  -    "          LSB
$B5  -  sign of mantissa
        Ø if mantissa = Ø
        + if mantissa  non-zero or plus
        - if mantissa  negative
```

The floating accumulator may be converted to a two byte integer in $B3 and $B4 (MSB, LSB) by a JSR $DØA7. On return to BASIC, an integer may be converted and passed in the floating accumulator. The MSB is loaded into the MOS 6502 accumulator A and the LSB into index register Y and then JSR $D278. Since the return address to BASIC is already on the stack and the integer-floating conversion might be the last step to execute, it is possible to do a JMP $D278 instead of a JSR $D278 and RTS.

Before executing USR from BASIC, locations 1 and 2 must be poked with the address, lo-hi, of the machine code subroutine. The address may be changed if the programmer desires to have more

than one routine resident at one time.

It is recommended that the machine language subroutines be located in protected areas of RAM such as the unused tape buffer.

example: floating point representation
1.5_{10}
8Ø CØ ØØ ØØ ØØ ØØ
$BØ →

USR function example #1

```
ØØØØ  4C  3A  Ø3          JMP   USR
                          INT = $B3
                           *  = $33A
Ø33A  2Ø  A7  DØ    USR   JSR  FLPINT
033D  A5  B3              LDA  INT
Ø33F  A6  B4              LDX  INT+1          Swap bytes
Ø341  85  B4              STA  INT+1            to use
Ø343  86  B3              STX  INT              as address
Ø345  A2  ØØ              LDX  #Ø               indirect
Ø347  A1  B3              LDA  (INT,X)          load
Ø349  A8                  TAY                   LSB in Y
Ø34A  8A                  TXA                   MSB in A
Ø34B  4C  78  D2          JMP  INTFLP
                          INTFLP = $D278
                          FLPINT = $DØA7
```

USR function example:

```
        X = USR(I)
-32768 ≤ I ≤ 32767
    Ø ≤ X ≤ 255
```

Returns the contents of the byte whose address is specified by I. The variable I is preserved. Parameter is passed in the floating accumulator and translation is performed by appropriate BASIC subroutines.

```
10000 DATA 32,167,2Ø8,165,179,166,18Ø,133
10100 DATA 18Ø,134,179,162,Ø,161,179,168
10200 DATA 138,76,12Ø,21Ø
10300 FOR I = 826 TO 845
10400 READ N:POKE I,N
10500 NEXT
10600 POKE 1,58
10700 POKE 2,3
```

This is a BASIC program to POKE the USR machine language subroutine from the previous example into the memory. The hex codes have been translated into decimal and placed in data statements. The memory region used is the 2nd cassette data buffer area. Note locations 1 and 2 are poked with the start address of the subroutine:

3*256+3*16+1Ø = 826

USR function example #2

```
Ø33A   2Ø   A7   DØ   LOGB2  JSR   $DØA7     floating to integer
Ø33D   AØ   ØØ             LDY   #Ø        LSB of result in
Ø33F   A5   B4             LDA   $B4       LSB of integer
Ø341   6A              SHIFT ROR   A
Ø342   9Ø   Ø5             BCC   DONE      switch closed
Ø344   C8                  INY
Ø345   CØ   Ø8             CPY    #8       no switches ?
Ø347   DØ   F8             BVE    SHIFT
Ø348   A9   ØØ         DONE LDA   #Ø        MSB in A = Ø
Ø34A   4C   78   D2        JMP   $D278     integer to floating
                           *=Ø
ØØØØ   4C   3A   Ø3        JMP   LOGB2     vector for USR
```

```
10 PRINT USR(PEEK(59471)):GOTO 1Ø
```

Switches connected to USR port can be wired to cause a low logic level. The port can be PEEK'ed and this routine returns the bit #(0-7) or 8 if no switch is closed.

commodore

COMMODORE BUSINESS MACHINES, INC.
901 CALIFORNIA AVENUE
PALO ALTO, CALIFORNIA 94304
TELEPHONE: (415) 326-4000 TELEX: 345-569
CABLE ADDRESS COMBUSMAC PLA

PIN OUT INFORMATION

USER PORT

PIN #	LABEL	DESCRIPTION
1.	Ground	Digital Ground
2.	T. V. Video	Video output used for external Display, used in diagnostic routine for verifying the video circuit to the display board.
3.	IEEE SRQ	Service request is used by a device to indicate the need for attention or service and to request an interruption of the current sequence of events. Is used in verifying operation of the SRQ in the diagnostic routine.
4.	IEEE EOI	Is used to indicate the end of a multiple byte transfer sequence. This pin verifies the EOI function when running the diagnostic routine.
5.	Diagnostic Sense	When this pin is low system power up the PET software jumps to the diagnostic routine rather than the BASIC routine.
6.	Tape #1 READ	Used with the diagnostic routine to verify cassette tape #1 read function.
7.	Tape #2 READ	Same as cassette #1 except this pin is for cassette #2.
8.	Tape Write	Used inconjunction with the diagnostic routine to verify operation of the WRITE function of both cassette ports.
9.	T. V. Vertical	T. V. vertical output for external display device. Verified in diagnostic.
10.	T. V. Horizontal	T. V. horizontal output for external display device. Verified in diagnostic.
11, 12,A	GND	Digital ground.
B	CAI	Is an interrupt and flag input, only from peripherals. (I.e. Handshake for data on PA Port)
C	PAØ	Input/Output lines to peripherals, and can be programmed independent of each other for input or output.
D	PAI	

PIN #	LABEL	DESCRIPTION
E	PA2	
F	PA3	
H	PA4	
J	PA5	
K	PA6	
L	PA7	
M	CB2	Can act as a totally independent interrupt as a peripheral control output or a serial input or output.
N	GND	Digital ground.

Cassete #2 Interface

A-1	GND	Power ground.
B-2	+5	Positive 5 volts to operate cassette circuitry.
C-3	Motor	Unregalated positive 6 volts to operate cassette motor.
D-4	Read	Read line from cassette.
E-5	Write	Write line to cassette, puts information on tape.
F-6	Sense	Senses. Closure of mechanical switch on cassette when motor is engaged.

Memory Expansion Port

All odd pins are grounded. (top side of board)

2B A0	Buffered	Address Bit 0 used for memory expansion
4B A1	Buffered	Address Bit 1 used for memory expansion
6B A2	Buffered	Address Bit 2 used for memory expansion

8B	A3	Buffered	Address Bit 3, used for memory expansion.
1Ø	BA4	Buffered	Address Bit 4, used for memory expansion.
12	BA5	Buffered	Address Bit 5, used for memory expansion.
14	BA6	Buffered	Address Bit 6, used for memory expansion.
16	BA7	Buffered	Address Bit 7, used for memory expansion.
18	BA8	Buffered	Address Bit 8, used for memory expansion.
20	BA9	Buffered	Address Bit 9, used for memory expansion.
22	BA1Ø	Buffered	Address Bit 1Ø, used for memory expansion.
24	BA11	Buffered	Address Bit 11, used for memory expansion.
26	NC		No connection
28	NC		No connection
30	NC		No connection
32	NS1	Active low	Address select for locations 1000-1FFF
34	NS2	Active low	Address select for locations 2000-2FFF
36	NS3	Active low	Address select for locations-3000-3FFF
38	NS4	Active low	Address select for locations 4000-4FFF
40	NS5	Active low	Address select for locations-5000-5FFF
42	NS6	Active low	Address select for locations 6000-6FFF
44	NS7	Active low	Address select for locations 7000-7FFF
46	NS9	Active low	Address select for locations 9000-9FFF
48	NSA	Active low	Address select for locations A000-AFFF
50	NSB	Active low	Address select for locations B000-BFFF
52	NC		No connection
54	$\overline{RES}$		Resets microprocessor.
56	IRQ		Interrupt request line to the microprocessor.
58	BØ2		Buffered phase 2 clock.
6Ø	BR/W		Buffered read or write enable.
62	NC		No connection.
64	NC		No Connection

66	BDØ	Buffered date bit Ø
68	BD1	Buffered date bit 1
70	BD2	Buffered date bit 2
72	BD3	Buffered date bit 3
74	BD4	Buffered date bit 4
76	BD5	Buffered date bit 5
78	BD6	Buffered date bit 6
80	BD7	Buffered date bit 7

IEEE-488 INTERFACE

1	DIO1	Data input/output bit Ø
2.	DIO2	Data input/output bit 1
3.	DIO3	Data input/output bit 3
4.	DIO4	Data input/output bit 3

The Transfer Bus

A handshake sequence is executed by the talker and the listeners over the Transfer Bus time a data byte is transferred over the Data Bus. The transfer Bus signal lines are defined as follows:

	Signal	Definition
7.	Not Ready for Data (NRFD)	An active low NRFD signal line indicates that one or more assigned listeners are not ready to receive the next data byte. When all of the assigned listeners for a particular data transfer have released NRFD, the NRFD line goes inactive high. This tells the talker to place the next data byte on the Data Bus.
6.	Data Valid (DAV)	The DAV line is activated by the talker shortly after the talker places a valid data byte on the Data Bus. An active low DAV signal tells each listener to capture the data byte presently on the Data Bus. The talker is inhibited from activating DAV when NRFD is active low.

	Signal	Definition
8.	Not Data ~~Data Not~~ Accepted (NDAC)	The NDAC signal line is held active low by each listener until the listener captures the data byte, NDAC goes inactive high. This tells the talker to take the byte off the Data Bus.

Management Bus

The Management Bus is a group of signal lines which are used to control data transfers over the Data Bus. The signal definitions for the Management Bus are as follows:

	Signal	Definition
9.	Interface Clear (IFC)	The IFC signal line is activated by the PET when it wants to place all interface circuitry in a predetermined quiescent state.
10.	Service Request (SRQ)	Any peripheral device can request the attention of the PET by setting SRQ active low. The PET responds by setting ATN active low and executing a serial poll to see which device is requesting service.
5.	EOI	Is used to indicate the end of a multiple byte transfer sequence.
11.	Attention (ATN)	This signal is activated by the PET when peripheral devices are being assigned as listeners and talkers. Only peripheral addresses and control messages can be transferred over the Data Bus when ATN is active low. After ATN goes high, only those peripheral devices which are assigned as listeners and talkers can take part in the data transfer.
12.	Chasis ground	Ground line to ground the chasis together.
A.	DIO5	Data input/output bit 4.
B.	DIO6	Data input/output bit 5.
C.	DIO7	Data input/output bit 6.
D.	DIO8	Data input/output bit 7.
E.	N GND	Digital grounds.

Connectors

Connector	J1	Display connector
	J2	Keyboard connector
	J3	Cassette #1 connector
	J4	Memory expansion connector
	J5	User Port connector
	J6	Cassette #2 connector
	J7	IEEE-488 connector

PIA	6520	UG8	
PIAL		$E810	59408
PIAL1		$E811	59409
PIAK		$E812	59210
PIAS		$E813	59411
PIA	6520	UB8	
IEE1		$E820	59424
IEEIS		$E821	59425
IEEO		$E822	59426
IEEOS		$E823	59427
VIA	6522	VA5	
PIA		$ E840	59456
SYNC		$ E841	59457
P2DB		$ E842	59458
P2DA		$ E843	59459
TIL		$ E844	59460
TIH		$ E845	59461
TILL		$ E846	59462
TILH		$ E847	59463
T2L		$ E848	59464
T2H		$ E849	59465
SR		$ E84A	59466
ACR		$ E84B	59467
PCR		$ E84C	59468
IFR		$ E84D	59469
IER		$ E84E	59470
SYNC1		$ E84F	59471

SYMBOLIC NAME	HEX ADDRESS	DECIMAL

PIAL

Ø	O	A
1	O	B keyboard decode
2	O	C
3	O	D
4	I	#1 Cassette on switch
5	I	#2 Cassette on switch
6	I	EOI input from IEEE 488
7	I	Diagnostic jumper sense

PIAL1

3	O	Blank to TV display

PIAK

Ø	I	
1	I	
2	I	
3	I	INput from
4	I	keyboard
5	I	scan
6	I	
7	I	

PIAS

3	O	#1 cassette motor control

		N
		O
		I
	N	T
	O	P
	I	I
	T	R
	C	C
	E	S
T	R	E
I	I	D
B	D	

IEE1

0	I	DI1	
1	I	DI2	
2	I	DI3	
3	I	DI4	in from IEEE
4	I	DI5	data lines
5	I	DI6	
6	I	DI7	
7	I	DI8	

IEEIS

3	0	$\overline{\text{NDAC}}$	to IEEE

IEEO

0	0	DO1	
1	0	DO2	
2	0	DO3	
3	0	DO4	Out to IEEE
4	0	DO5	data lines
5	0	DO6	
6	0	DO7	
7	0	DO8	

IEEOS

3	0	$\overline{\text{DAV}}$	to IEEE

SYNC	or	SYNC1	(See 6522 spec)
0		I/O	
1		I/O	
2		I/O	
3		I/O	
4		I/O	User port
5		I/O	
6		I/O	
7		I/O	

PIA

0	I	NDAC
1	O	NRFD
2	O	ATN
3	O	Cassette write
4	O	#2 cassette motor
5	I	Display on (sync)
6	I	NRFD
7	I	DAV

PCR

3	O	Bus or graphics character set

ACR

3	I/O	User port serial line.

I/O checkout
JF 10-28-77

1 UG8 - keyboard test

a. plug in a working keyboard

b. type the following sequence of characters and verify that they appear on the screen:

```
! " # $ % 1 & / ( ) ←
Q W E RT YU I O P ↑
A S D F G H J K L :
Z X C V B N M , ; ?
@ [ ] < >
7 8 9 / 456 * 1 2 3 + 0 . - =
```

[SPACE] [RETURN]

PET should respond with

? SYNTAX ERROR
READY.

c. type this line

? " [CLR HOME] [CURSOR ↓] [CURSOR →] [OFF RUS] AAA [RETURN]
SHIFT

Screen should clear and

[AAA]

READY.

be printed

d. if keyboard does not work

i recheck UG8 for bent pins

ii replace UG8

iii inspect for shorts near

UG9 - UG8 -J5

2. UG8 - UA5- Cassette test

a. built in cassette motor should be off.

b. verify cassette motor operation by pressing PLAY , REWIND , and FAST FORWARD

c. if built in cassette problem

i recheck UG8 for bent pins

ii replace UG8

iii check Q1 - Q6

d. plug in second cassette

e. verify cassette motor operation Motor should only run when PLAY , REWIND , & FAST FORWARD are pressed.

f. if 2nd cassette problem then go to step C and check UA5

3. CA2 UB5 TEST

a. type

POKE 59409 , 52

the TV display should go blank

b. reset the PET

c. type

ZZZ CURSOR

POKE 59468 , 14 R E T U R N

THE ROW OF CHARACTERS SHOULD CHANGE TO

zzzZZZ

4. UB8 - UA7 - UA9

a) Using a voltmeter on 5-10 volt range check pins on J1 to make sure only 1 pin at a time is high. Black wire to ground. Red to test pins 1,2,3,4,13,14,15,16

POKE	59426,	1	only pin	1	high
	"	2	"	2	"
		4		3	
	"	8	"	4	
		16		13	"
		32		14	
	"	64	"	15	"
		128		16	

b) Type in this program:

```
1Ø   POKE 59426, 255
2Ø   ? PEEK (59424)  :  GOTO 2Ø
RUN
```

ground this pin

on J1	See this number on screen
1	254
2	253
3	251
4	247
13	239
14	223
15	191
16	127

5. User port test UA5

a) Reset machine

b) type in this program

```
1Ø ? PEEK (59471) : GOTO 1Ø
RUN
```

ground this pin on J2	See this number on screen
C	254
D	253
E	251
F	247
H	239
J	223
K	191
L	127

c) Attach black lead of voltmeter to ground. Test pins on J2:

C,D,E,F,H,J,K,L

type

POKE 59471 ,	1	only	C	high
"	2	"	D	"
	4		E	
	8		F	
"	16	"	H	"
	32		J	
"	64	"	K	"
	128		L	

commodore

COMMODORE BUSINESS MACHINES, INC.
901 CALIFORNIA AVENUE
PALO ALTO, CALIFORNIA 94304
TELEPHONE: (415) 326-4000 TELEX: 345-569
CABLE ADDRESS COMBUSMAC PLA

PET EDITING

When you press one of the PET's cursor control keys, you may be in one of two editing modes.

1. DIRECT CURSOR CONTROL

   The cursor is moved as soon as you press the cursor control key.

2. PROGRAMMED CURSOR CONTROL

   The cursor movement is executed during a program run. It is part of a PRINT statement and has been enclosed within quotation marks.

FUNCTION	KEYS TO PRESS		ASCII	CHARACTER
CURSOR UP	SHIFT	CRSR ↑↓	145	●
CURSOR DOWN		CRSR ↑↓	17	Q
CURSOR LEFT	SHIFT	CRSR ⇐⇒	157	▌
CURSOR RIGHT		CRSR ⇐⇒	29	]
CLEAR SCREEN	SHIFT	CLR HOME	147	♥

FUNCTION	KEYS TO PRESS		ASCII	CHARACTER
HOME CURSOR		CLR HOME	19	S
INSERT CHARACTER *	SHIFT	INST DEL	148	
DELETE CHARACTER *		INST DEL	20	T
REVERSE FIELD		OFF RVS	18	R
RESET REVERSE	SHIFT	OFF RVS	146	

* The INSERT and DELETE functions are not programmable.

Use CHR$ (20) to delete during program run and CHR$ (148) to insert during program run.

PET uses the quotation mark to signal the beginning of a string literal, as in a DATA or PRINT statement. When attempting to edit a program line, the User should be aware that if PET sees an opening quote, it will consider all cursor movement instructions as part of the string.

DIRECT CURSOR CONTROL

In DIRECT mode, the User is creating program code. The cursor control keys allow the User to insert or delete characters at will unless he specifically indicates (by typing a quotation mark) that the cursor movement is to be a part of the created code.

When entering program code, the User can correct typographic errors in one of four ways.

A) Delete all characters back to the error, then retype.

B) If no quotation marks have been used, backspace (cursor left) over the intervening characters until the cursor is positioned over the error, retype the character, then forward space (cursor right) to the next desired character position to be typed.

C) If a quotation mark has been used, press [RETURN] to leave the program line. Then move the cursor up and over to one space past the error. Press [INST DEL] to delete the error, press [SHIFT] and [INST DEL] to create an opening, and type in the correct character, then forward space to the next desired character position to be typed.

Programmed cursor control is no longer in effect.

D) Another method is to close the quotes (type the ending quotation mark) then backspace to the offending character and retype. Again, programmed cursor control is no longer in effect.

There may be occasions when it is appropriate to lengthen a statement line. If the cursor is moved to the end of an existing line, the additional characters may be typed in. The cursor will wrap around to the next lower line if more than 40 positions are used. If the lower line contains a program statement, it can be over-typed. Extra characters remaining from that previously typed line must be deleted or they will be incorporated into the line being edited.

Original Program

```
10 PRINT " NOW  IS   THE   TIME   FOR  ALL"
20 PRINT  "THE END"
```

Move the cursor intil it is positioned over the closing quotes in statement 10, and type GOOD MEN TO COME"

```
10 PRINT "NOW  IS   THE   TIME  FOR  ALL  GOOD  M
EN   TO   COME"   THE END"
```

delete THE END" by spacing over the characters, using the SPACE bar.

Now LIST

```
10 PRINT "NOW IS THE TIME FOR ALL GOOD M
EN TO COME"
20 PRINT "THE END"
```

If you wish to insert characters within a statement line, position the cursor over the first character to be shifted to the right, press [INST DEL] with the [SHIFT] key. If the new spaces increase line length to greater than 40 spaces, a space will open up between the line being edited and the next program line, and the characters to the right of the insertion will move into the opened space. This is difficult to show on paper, so just follow the instructions and watch the result on your screen.

1. Type this program

   ```
   10 PRINT "NOW IS THE TIME TO COME"
   20 PRINT "THE END"
   ```

2. List the program
3. Move the cursor to the letter T in the word TO in statement 10.
4. Hold the [SHIFT] key and press the [INST DEL] key 18 times

   (Here's where the screen will show a space being opened between statement lines)
5. Type FOR ALL GOOD MEN T
6. Press [RETURN]
7. LIST the program again

Using Direct Cursor Control while coding a string literal:

To edit a string literal, such as a print message or a data statement, the user must press the [RETURN] key and leave the statement line. A literal cannot be edited (except for character deletion and retyping) while it is being originated, because all cursor controls except delete and insert are programmable. The user must leave the statement line via a carriage return, then move the cursor back to the offending character and retype. Furthermore, to program cursor controls within the string after having left the line, the user must use the INSERT function to open up spaces into which he can then type the appropriate control character.

The user can, of course, close the quotes, and thereby signal PET that he is through with the literal message. However, once the second quote mark has been typed, PET will no longer recognize cursor movement as a part of created code, and the cursor will move according to the function represented by the key pressed.

PROGRAMMED

EDIT FUNCTIONS

The User can control the position of the cursor on the screen in order to PRINT in a specific position. For example:

10 PRINT "♥" Clear screen

20 FOR I = 1 TO 10

30 Print "Q" Cursor down

```
40   NEXT  I

50   FOR J = I  TO  10

60   PRINT "[J]";                          Cursor right

70   NEXT  J

80   PRINT  "HI"
```

Will PRINT the word "HI" in column 11 on LINE 11. This program can be more simply written.

```
10   PRINT"[♥]";:FOR  I = 1  TO  10:Print"[Q]":Next

20   FOR  J = 1  TO  10:Print"[J]";: NEXT:Print"Hi"
```

OR, even simpler using a single PRINT statement:

```
10   PRINT  "♥ Q Q Q Q Q Q Q Q Q Q ] ] ] ] ] ] ] ] ] ] HI"
```

A SHORT DESCRIPTION OF THE IEEE-488 BUSS FOR THE PET

This description covers the pin-out and signal designations for the IEEE-488 BUS as implemented on the PET. A brief description of the PET BASIC commands for the IEEE-488 BUSS is also included.

I. INTERCONNECTION

The PC card edge on the left-rear of the PET labeled J1 has the IEEE-488 signals. For reasons of economy, a standard IEEE-488 connector is not included.

A standard 12-position, 24-contact edge connector with .156" spacing is attached to the PET PC card. Some typical connectors and part numbers are:

EDGE CONNECTOR BRAND	PART #
SYLVANIA	6AGØ1-12-1A1-Ø1
AMP	53Ø657-3
AMP	53Ø658-3
AMP	53Ø654-3
CINCH	251-12-9Ø-16Ø

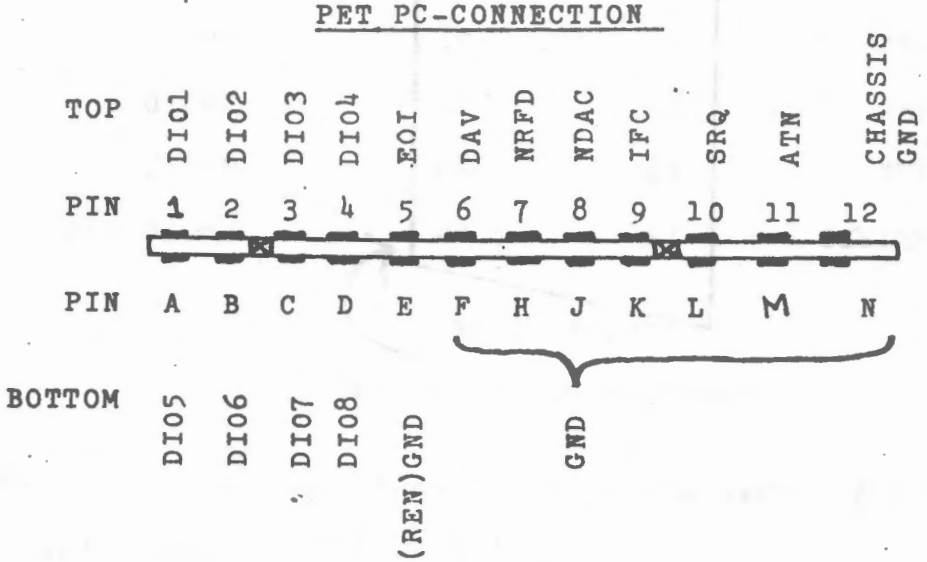
In a pinch, a larger edge connector (such as 15 or 22 positions) can be cut with a hacksaw to provide a temporary substitute.

The IEEE-488 (or HP-GB/IB) connector is available from:

IEEE CONNECTOR BRAND	PART #
CINCH	571Ø24Ø SOLDER PLUG
CINCH	572Ø24Ø " RECEPTACLE
AMP	5523Ø1-1 INSULATION DISP PLUG
AMP	5523Ø5-1 " " RECPT

The pin designations and numbers are identical for both connectors. A short cable (i.e., 15 conductor ribbon, etc.) may be used to join the connectors.

PET PC-CONNECTION



Polarization slots between 2-3 and 9-10

IEEE-488 CONNECTION

IEEE DESIGNATION	PIN		IEEE DESIGNATION
DIO1	1	13	DIO5
DIO2	2	14	DIO6
DIO3	3	15	DIO7
DIO4	4	16	DIO8
EOI	5	17	REN
DAV	6	18	GND6
NRFD	7	19	GND7
NDA	8	20	GND8
IFL	9	21	GND9
SRQ	10	22	GND10
ATN	11	23	GND11
SHIELD	12	24	LOGIC GND

SCHEMATIC OF

CONNECTOR POLARIZATION

SUGGESTION: When wiring the edge-connector to the IEEE connector, include a 16 pin DIP socket to jumper the control lines. This permits easy modification of the connection to the PET to handle some non-standard attributes of the PET's IEEE-488 Interface. (These are described later.)

II. SOME PHYSICAL LIMITATIONS:

1. Maximum length: 20 meters

II. SOME PHYSICAL LIMITATIONS: (cont'd)

2. Maximum inter-device spacing: 5 meters
3. Maximum number of devices: 15
4. Maximum data rate: 250 KHZ (1 MHZ with tristate drivers)

III. GENERAL CONCEPTS

The IEEE-488 BUS is comprised of three functional groups of lines:

DIO1 DIO2 DIO3 DIO4 DIO5 DIO7 DIO8	DATA BUSS	The data buss transfers data at a rate controlled by the slowest device on the buss. The form is byte-serial/bit-parallel (i.e., a byte at a time). Also transferred on the data buss are peripheral addresses or control information.
NRFD DAV NDAL	TRANSFER BUSS	This set of lines controls the transfer of data on the data buss. This buss ensures that data is valid and that all transfers are complete before new data is sent.
ATN SRQ IFL REN EOI	MANAGEMENT BUSS	The management buss controls the state of the buss, commands for the devices, etc.

The buss can support three classes of devices:

1. TALKERS. At any given time, only one device may transmit data to the buss. Devices capable of this are talkers.
2. LISTENERS. As many devices as required may receive data from the buss.
3. CONTROLLERS. At any moment, only one device may control the buss. Control can be passed to other devices capable of controlling the buss.

IV. BUSS SIGNALS

A. THE DATA BUSS

Lines DI01 - DI08 are the data buss. These are active-low bidirectional lines. (This means a line is normally high. Any device can ground the line, making a signal present.)

Data is transferred in bytes, one bit per line, with the MSB in DI08. The forms of data are:

1. Data from instruments
2. Address - primary or secondary
3. Control words

B. THE TRANSFER BUSS

The transfer of data over the data bus is controlled by these three lines. The handshake sequence ensures complete transmission and reception by the slowest device on the bus.

B. THE TRANSFER BUSS (cont'd)

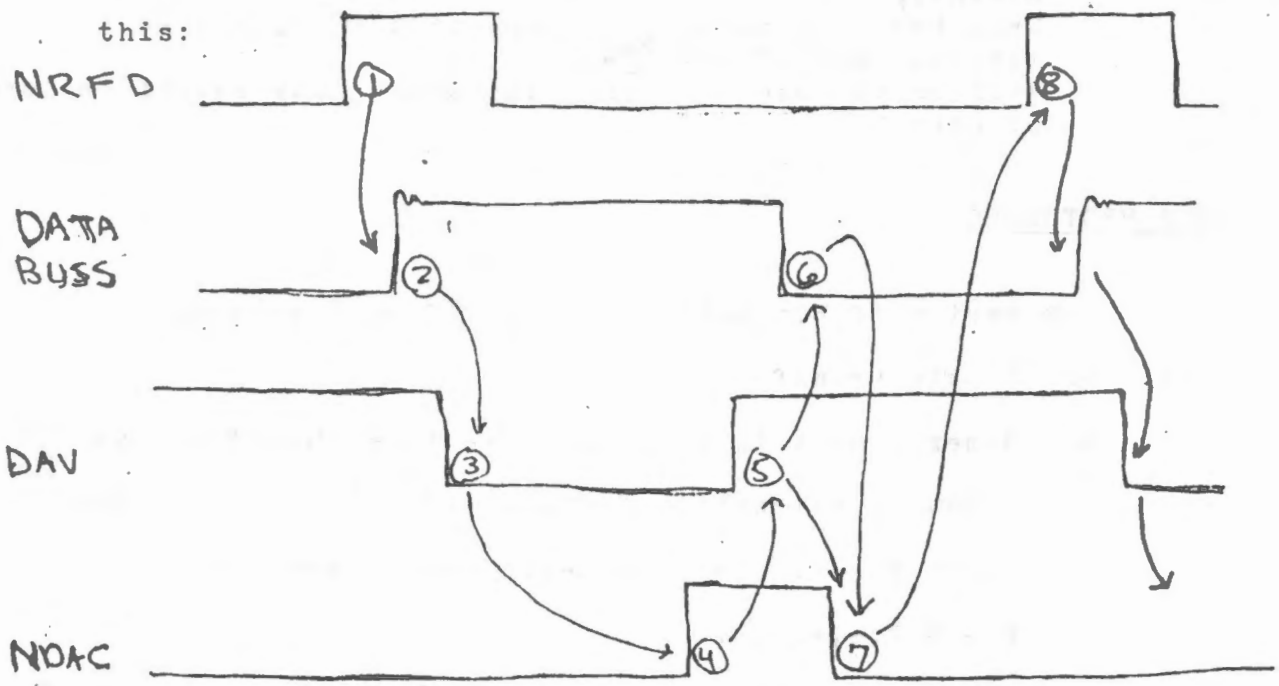
LINE

NRFD NOT READY FOR DATA. When this line is low, one or more listeners are not ready for the next byte of data. When all devices are ready, NRFD goes high. This informs the talker to put the next byte on the data buss.

DAV DATA VALID. When this line goes low, the listeners may read the data byte on the data buss. The talker cannot put DAV low if NRFD is low (All listeners must be ready first).

NDAC DATA NOT ACCEPTED. Each listener holds this line low until it has finished reading the data byte. When NDAC goes high, the talker can remove the data from the data buss and go to the next byte.

A simplified diagram of the handshake sequence looks like this:



EVENT

1. When NRFD goes high, the talker is permitted to put data on the data buss,
2. The data is put on the buss, and after a settling interval
3. DAV is set low to indicate data is valid. The devices accept data.
4. When all devices have accepted the data, NDAC goes high, permitting
5. The talker to remove DAV and
6. Take the data off the data buss.
7. The listeners note the removal of DAV and resets NDAC in preparation for the
8. Next data transfer cycle

NOTE: When PET is a listener, it expects DAV within 64 milliseconds of NRFD going low(1-3 within 6Ø milliseconds)
When PET is a talker, it expects NDAC within 64 milliseconds of DAV 3-4.
Failure to observe these limitations may result in loss of data.

DATA PROTOCOLS

1. Any series of bit patterns is valid on the buss.
2. ASCII Data transfer:
   a. Numeric data is transmittable in either floating point or scientific format, with most significant digit first. Valid numeric characters are: Ø - 9,E,e,+,-, .
   b. Strings are terminated with return or activation of the EOI line or both.

C. THE MANAGEMENT BUSS

Five signal lines control the activity of the buss and define the meaning of the data being transferred (data, address or control)

LINE

ATN ATTENTION. The controller sets this line to low when it is assigning devices as listeners and talkers. When ATN is low, only peripheral addresses and control messages are on the data buss. When ATN is high, only assigned devices can transfer data.

SRQ SERVICE REQUEST. Any device can set SRQ low to alert the controller that a device requires service. When the controller sets SRQ, it sets ATN low and does a "service poll" to find out which device wants service. NOTE: This bit is accessible in the PET. However, the PET 488 software does not include this function, and it is up to the user to do so.

IFC INTERFACE CLEAR. The controller sets this line to initialize the bus. NOTE: PET only activates this line when it is reset or powered up. The signal is low for about 100 milliseconds. If the user wants this function, it is suggested he place a switch on this line.

REN REMOTE ENABLE. Some devices hang the option of either operating from their front panels or the IEEE bus. When REN is low, control is via the bus. NOTE: The PET has this line set permanently low (the pin is grounded).

Put a switch in the line if REN control is desired.

EOI <u>END or IDENTIFY</u>. When a talker is finished with data transfer, it sets EOI low. (This is optional). The controller always sets EOI low when it is finished. (EOI is set low during last byte transferred)

V. PET COMMANDS/BASIC STATEMENTS PERTINENT TO IEEE-488 BUSS

It is assumed the user knows how to read and write data to the tape casette files. See the casette tutorial bulletin for coverage of this area.

The IEEE-488 buss appears as a file to BASIC. The following BASIC items are pertinent:

OPEN CLOSE	Open/Close files (assign devices)
PRINT# INPUT# GET#	Transfer data
CMD	Direct PET's output elsewheres
ST	I/O status variable

The following descriptions are only about the aspects which pertain to the buss.

<u>OPEN</u> [Logical Address], [Physical Device], [Secondary Address], "Filename"

The <u>Logical Address</u> is 1-255 and is referenced by the CLOSE, PRINT#, INPUT#, AND GET# statements.

The Physical Device is the Primary Device Address, and the range is 4-15.

The Secondary Address is optional. If omitted, none is sent. The range is Ø-31. Bits 6 and 7 are set when sent to the bus.

If the address was 2, ØØØØØ1Ø
is sent as Ø11ØØØ1Ø

The Secondary Address is sent only on execution of the OPEN and CLOSE statements.

A specific form of the Secondary Address is sent if a Filename is specified for OPEN and CLOSE. Bit 8 is set in both cases, and bit 5 set on OPEN. As bit 5 is used to specify a control command,

SA 0-15 are files
16-31 are commands

CLOSE This will send the Secondary Address (if any) to the device specified by the open command.

PRINT# This will send ASCII characters to the IEEE-488 bus. If it is desired to set the most significant bit, use variations of:

PRINT# 2, CHR$ (X) X range: Ø - 255

INPUT# Receives characters according to BASIC INPUT rules.

GET# Gets a character or a digit.

NOTE: PRINT, INPUT, & GET all refer to the Logical Address specified in the OPEN statement.

CMD LOGICAL ADDRESS All BASIC output is now sent to the device specified by a prior OPEN statement or command. This has two useful properties:

1. BASIC programs can be listed to a file or device.
2. CMD leaves the IEEE bus active, permitting more than one listener on the IEEE bus.

NOTE: Each time a PRINT# statement is executed, the following sequence happens:

1. The device specified in the corresponding OPEN statement is designated a listener.
2. The data is sent.
3. All devices are set to "not listen" status (UNL).

A similar sequence is used for INPUT#, with designation of a talker, and an untalk (UNT) command.

If a CMD is executed first, the specified device will also be able to listen when the PRINT# is executed. Note that CMD must be executed again if more than one PRINT# statement is used for multiple devices as PRINT #s.

ST STATUS WORD. The following bits in the BASIC variable, ST, pertain to the IEEE-488 bus:

BIT	AND MASK	
Ø	1	Time out on data transfer
1	2	.... read error
....		
6	64	EOI

ST STATUS WORD. (cont'd)

BIT	AND MASK	
7	128	Device not present

Use the form: IF (ST) AND MASK THEN ---(1, 2, 64, or 128) to detect these conditions. The test should be done immediately after the I/O operation of interest.

TIME OUT. BIT 4 MASK: 1 The IEEE device has not responded within 65 milliseconds (time out interval).

READ ERROR. BIT 1 MASK: 2 The IEEE device has not provided DAV within the time out - INPUT# or GET#.

EOI. This is set when an IEEE device finishes transmission of data (see the manual for the instrument as some devices won't do this). A convenience feature!

DEVICE NOT PRESENT. When I/O is initiated, the device did not respond to its physical address. This generates an error message and returns you to BASIC command level.

VI. IEEE-488 REGISTER ADDRESSES

If you are bold, here are the IEEE-488 hardware addresses for the PET. Attempting to control the bus via peek and poke will probably fail as the timeouts for the 488 devices may be exceeded.*
Happy hacking!

*Use machine language.

VI. IEEE-488 REGISTER ADDRESSES (cont'd)

NAME	HEX ADDRESS	DECIMAL ADDR	BITS	IEEE LINES	
IEEI	$E82Ø	59424	Ø-7	DIO 1-8	(INPUT)
IEE	$E822	59426	Ø-7	DIO 1-8	(OUTPUT)
IEEIS	$E821	59425	3	$\overline{NDAC}$	(OUTPUT)
IEEOS	$E823	59427	3	$\overline{DAV}$	(INPUT)
			4	SRQ	**
PIAL	$E81Ø	59408	6	$\overline{EOI}$	(INPUT)
PIA	$E84Ø	59456	Ø	$\overline{NDAC}$	(INPUT)
			1	$\overline{NRFD}$	(OUTPUT)
			2	$\overline{ATN}$	(OUTPUT)
			6	$\overline{NRFD}$	(INPUT)
			7	$\overline{DAV}$	(OUTPUT)

** CB1 input of VIA 6522 (see MOS Technology 6522 specification).

Good luck! Let us know if you do anything interesting.

This bulletin prepared by Gregory Yob,
Software Editor, Commodore
Business Machines

A LIST OF IEEE-488 DEVICES TO USE WITH PET

You can get the IEEE-488 specs by sending $10.00 plus postage and handling to:-

IEEE SERVICE CENTER
445 HOES LANE
PISCATAWAY, NJ 08854

While we list an RS232/IEEE-488 Interface, it really doesn't exist yet. R. Bailey Associates of 31 Bassett Road, London NW10 however do make such an Interface and you are advised to write to them for price and delivery.

IEC/IEEE PRODUCT INTRODUCTIONS

66(C.O.) 22, IEEE 488, ANSI MC1.1 COMPATIBLE

5500B	UNIVERSAL COUNTER TIMER	BALLANTINE
76A	AUTOMATIC CAPACITANCE BRIDGE	BOONTON ELECTRONICS CORP.
3347	AUDIO FREQUENCY ANALYZER	BRUEL & KJAER
4426	NOISE LEVEL ANALYZER	" "
1554	STRAIN INDICATOR	" "
	BUS CABLE ASSEMBLY	BUNKER-RAMO CORP.
DSM44	DIGITAL MULTIMETER	CALIFORNIA INSTRUMENT CO.
	BUS CABLE ASSEMBLY	COMPONENT MFG. SERVICES
340	MATERIALS TESTING FUNCTION GENERATOR	
605-145	WAVEFORM GENERATOR, ASCII PROGRAMMER	DANA EXACT ELECTRONICS INC.
801	FREQUENCY SYNTHESIZER	" " " "
802	" "	
55	MICROPROCESSING GPIB (5000, 5900, 6900 DVMS)	
9015	MICROPROCESSING TIME/COUNTER	DANA LABS INC.
9035	" " "	
7500	DIGITAL MULTIMETER	DATA PRECISION CORP.
101	UNIVERSAL TIMER/COUNTER	
103	" " "	
105	" " "	
111	DIGITAL FREQUENCY COUNTERS	DATA TECHNOLOGY (RACAL)

113	DIGITAL FREQUENCY COUNTERS	DATA TECHNOLOGY (RACAL)
115	" " "	
117	" " "	
4880	BUS INTERFACE COUPLER	DATA WORKS INSTRUMENTATION
3000	HF COMMUNICATIONS RECEIVER	DECCA COMMUNICATIONS LTD.
IEC11-A	CONTROLLER (PDP-11)	DIGITAL EQUIPMENT CORP.
1015A	9 TRACK TAPE	DYLON CORPORATION
1015PE	" " "	" "
1015B	7 TRACK TAPE	
331	MICROWAVE COUNTER	EIP EXACT
351D	COUNTER	
451	MICROWAVE PULSE COUNTER	
296	AUTOMATIC LRC DIGITAL METER	ELECTRO SCIENTIFIC INDUSTRIES
501J	PROGRAMMABLE VOLTAGE STANDARD	ELECTRONIC DEVELOPMENT CORP.
	PROGRAMMABLE OSCILLATOR (A.C. POWER)	ELGAR CORPORATION
9880	INTERFACE COUPLER	FAIRCHILD INSTRUMENTATION SYSTEMS
FF303	ATE SYSTEM	FAULTFINDERS INC.
1953A	UNIVERSAL COUNTER-TIMER	
6010A	SYNTHESIZED SIGNAL GENERATOR	
6011A	SIGNAL GENERATOR	FLUKE MFG. CO.
8500	SYSTEMS MULTIMETER	

1792	LOGIC TEST SYSTEM (I/O PORT)	GENRAD
436A	POWER METER	HEWLETT-PACKARD PRODUCTS
3320B	FREQUENCY SYNTHESIZER (11235A)	"
3330B	AUTOMATIC SYNTHESIZER/SWEEPER (11235A)	"
3455	VOLTMETER	"
3490A	DIGITAL MULTIMETER	"
3495A	SCANNER	"
3571A	TRACKING SPECTRUM ANALYZER	"
3745A	SELECTIVE LEVEL MEASURING SET	"
3964A	INSTRUMENTATION TAPE RECORDER	"
3968A	INSTRUMENTATION TAPE RECORDER	"
4261A	LCR METER (OPT. 101)	"
5150A	THERMAL PRINTER	"
5312A	INTERFACE MODULE (5300B MEASURING SYSTEM)	"
5328A	UNIVERSAL COUNTER	"
5340A	FREQUENCY COUNTER	"
5341A	FREQUENCY COUNTER	"
5345A	ELECTRONIC COUNTER	"
5353A	FREQUENCY COUNTERS, CHANNEL PLUG-IN	"
5363A	TIME INTERVAL PROBES	"
5354A	CONVERTER PLUG-IN	
5942A	TRANSMISSION IMPAIRMENT MEASURING SET	"
8016A	WORD GENERATOR	"
8503A	AUTOMATIC RF NETWORK ANALYZER	"
8505A	AUTOMATIC RF NETWORK ANALYZER	"
8620C	MICROWAVE SWEEP OSCILLATOR	"
8660A/C	SYNTHESIZED SIGNAL GENERATOR	"
8672A	MICROWAVE SYNTHESIZER	"
9871A	IMPACT PRINTER	"
10745A	LASER TRANSDUCER SYSTEM COUPLER	"
47310A	A/D CONVERTER	"
59301A	ASCII PARALLEL CONVERTER	"
59303A	DIGITAL-TO-ANALOG CONVERTER	"
59304A	NUMERIC DISPLAY	"
59306A	RELAY ACTUATOR	"
59307A	DUAL VHF SWITCH	"
59308A	TIMING GENERATOR	"
59309A	DIGITAL CLOCK	"
59310A/B	21MX COMPUTER INTERFACE	"
59401A	BUS SYSTEM ANALYZER	"
59403A	HP-IB COMMON CARRIER INTERFACE	"
59405A	9820, 9830, CALCULATOR INTERFACE	"
59500A	MULTIPROGRAMMER (6940B) INTERFACE	"
98034A	HP-IB I/O (9825A)	"
98135A	HP-IB I/O (9815A)	"

3050B	DATA ACQUISITION SYSTEM	HEWLETT-PACKARD PRODUCTS
3042A	NETWORK ANALYZER SYSTEM	"
3044A	SPECTRUM ANALYZER SYSTEM	"
3045A	SPECTRUM ANALYZER SYSTEM	"
8507A	NETWORK ANALYZER SYSTEM	"
DTS70	DIGITAL TEST SYSTEM (PORT)	"
8580B	AUTOMATED SPECTRUM ANALYZER (PORT)	"
9500D	AUTOMATIC TEST SYSTEM (PORT)	"
RS432	MICROPROCESSOR DATA & TIMING GENERATOR	"
RS648	TIMING SIMULATOR/WORD GENERATOR	
IM5200	FPLA LOGIC ARRAY	INTERSIL
SPG-800	SIGNAL GENERATOR	INTERSTATE ELECTRONICS
395	LOCK-IN ANALYZER	ITHACO
7802-ISB	SYSTEM 1 (I/O PORT) (5900.6900. DMM VIA MICROPROCESSING GPIB)	KEITHLY INSTRUMENTS INC.
SN-488	POWER SUPPLY	KEPCO
	RELAY DRIVER	MICROCOMPUTER ASSOC
	RS-232/IEEE-488 INTERFACE	"
1180	DATA ACQUISITION & PROCESSING SYSTEM	NICOLET INSTRUMENT CORP.
PM2441	DIGITAL VOLTMETER	
PM2460	SCANNER	
PM2467	DIGITAL VOLTMETER	N.V. PHILLIPS
PM2527	PRINTER	
PM6625	COUNTER	
PM6650	COUNTER	
	S 100 to IEEE	PICKLES & TROUT
4001	PROGRAMMABLE LOW-PASS FILTER	PRECISION FILTERS INC.
488	FLEXIBLE CARTRIDGE DISC SYSTEM	PROCESS DYNAMICS INC.
FFT/S15	REAL TIME SPECTRUM ANALYZER	ROCKLAND SYSTEMS CORP.
PCL/PCW	CARD READER/CODE CONVERTER (SMU, SMDV, DPVP)	RHODE & SCHWARZ
SMPU	RADIO SET TEST ASSEMBLY	

2017	UNIVERSAL COUNTER	
2711	UNIVERSAL COUNTER	SCHLUMBERGER
6054B/C	MICROWAVE COUNTERS	
6063	AUTOMATIC COUNTER	
7115	DIGITAL MULTIMETER	
DPSD-50	DIGITAL POWER SOURCE	SYSTRON-DONNER
1600	MICROWAVE SYNTHESIZER	
4051	GRAPHIC COMPUTING SYSTEM	
4662	DIGITAL PLOTTER	TEKTRONIX
4924	MAGNETIC TAPE UNIT	
1625	LOGIC ANALYZER	VECTOR ASSOC. INC.
2254	COMPUTER BASED CONTROLLER (2200)	WANG
152	FUNCTION GENERATOR	
158	WAVEFORM GENERATOR	WAVETEK
159	WAVEFORM GENERATOR	
172	PROGRAMMABLE SIGNAL SOURCE	
4311B	FREQUENCY/PHASE-LOCK MEAS. SYSTEM	WEINSCHEL ENGR.

commodore

COMMODORE BUSINESS MACHINES, INC.
901 CALIFORNIA AVENUE
PALO ALTO, CALIFORNIA 94304
TELEPHONE: (415) 326-4000 TELEX: 345-569
CABLE ADDRESS COMBUSMAC PLA

ERROR MESSAGES

When an error occurs, PET returns to Command level and displays READY on its TV screen. Variable values and the program text remain intact, but the program cannot be continued using the CONT command. GOSUB and all FOR...NEXT context is lost, insofar as the current run is concerned.

When an error occurs in a program statement, the error message display will indicate the line number in which the error occurred.

When the error occurs in a direct, or command level, statement, no line number is displayed with the error message.

Error Message	What caused the error and how to fix it
CAN'T CONTINUE	Attempt to continue a program when none exists, an error occurred, after a new line was typed into the program, or a correction was made to an existing line. Correct the error, then use a directed GOTO to get back into the program, or type RUN and start over.

Error Message	What caused the error and how to fix it
DIVISION BY ZERO	Dividing by zero is an error. Check the expression used for the denominator in the offending arithmetic statement, then correct it so it can never be evaluated as Ø.
ILLEGAL DIRECT	Use of an INPUT, GET, or DEF statement as a direct command. Avoid using these statements as direct commands.
ILLEGAL QUANTITY	The parameter passed to a math or string function was out of range. "ILLEGAL QUANTITY" errors can occur due to: a. a negative matrix subscript, such LET A (-1) = 0 b. an unreasonably large matrix subscript: >65535 c. LOG-negative or zero argument, as LOG(-X) d. SQR-negative argument, as SQR(-4)

Error Message	What caused the error and how to fix it
ILLEGAL QUANTITY (cont'd)	e. A↑B if A is a negative variable and B is not an integer. (It works if a constant is used instead of a variable; i.e. -4↑B, because exponentiation is performed before unary minus.) f. A call to USR before the address of the machine language subroutine has been patched in. Be sure the argument is within the range of the function being used. a. subscripts must be equal to or greater than Ø, and less than or equal to 256. b. LOG requires a non-negative argument.
NEXT WITHOUT FOR	The variable in a NEXT statement corresponds to no previously executed FOR statement. The FOR part of a FOR...NEXT loop must be inserted or the offending NEXT part of the loop must be deleted. Be sure the index variables are the same at both ends of the loop.

Error Messages	What caused the error and how to fix it
NEXT WITHOUT FOR (cont'd)	Example: FOR I= 1 TO 10 . . . . NEXT I
OUT OF DATA	A READ statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program. Use the RESTORE statement to restore the data so PET can read it again, or restrict the number of READs to the correct number of DATA elements, or add more DATA elements, or use a flag at end of data list - check for it before reading.
OVERFLOW	The result of a calculation was too large to be represented in BASIC's number format. (If an underflow occurs, zero is given as the result and execution continues without any error message being printed.)

Error Messages	What caused the error and how to fix it
OVERFLOW (cont'd)	You requested a number greater than even PET can remember. Try asking for a smaller number. The largest possible number is 1.70141183E+38. Change the order of your calculations.
REDIMENSIONED ARRAY	After a matrix was dimensioned, another dimension statement for the same matrix was encountered. This error often occurs if a matrix has been given the default dimension 10 because a statement like A(I)=3 is encountered and then later in the program, a DIM A(100) is found.
	Check to see if you have used a GOTO to branch back to a statement preceding the DIM statement, or see if the DIM statement is inside a FOR....NEXT loop or a subroutine that will be executed more than once, or if you have used an array element before using the DIM statement. Make DIM one of the first lines in your program.

Error Message	What caused the error and how to fix it
RETURN WITHOUT GOSUB	A RETURN statement was encountered without a previous GOSUB statement being executed. Either insert a GOSUB or delete the RETURN. Maybe you fell through the program and should enter an END statement before the first subroutine statement to prevent falling through.
STRING FORMULA TOO COMPLEX	A string expression was too complex. Break up the string into two or more shorter strings.
STRING TOO LONG	Attempt was made by use of the concatenation operator to create a string more than 255 characters long. A number is printed as SPACE-NUMBER-CURSOR RIGHT Break up the string into two or more shorter strings.
SUBSCRIPT OUT OF RANGE	An attempt was made to reference a matrix element which is outside the dimensions of the matrix.

Error Message	What caused the error and how to fix it
SUBSCRIPT OUT OF RANGE (cont'd)	This error can occur if the wrong number of dimensions are used in a matrix reference; for instance, LET A(1,1,1)=Z when A has been dimensioned DIM A(2,2). You must either increase the space you requested for the array (change a DIM A(10) to a DIM A(20), for example) or alter the number of dimensions you asked for (change from DIM A(10,10) to DIM A(10,10,10) or from DIM B(10,10,10) to DIM B(10,10) for example).
SYNTAX ERROR	Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc. This one is hard to find, but easy to fix. Examine the offending statement carefully and insert or delete whatever is necessary.
TYPE MISMATCH	The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa; or a function which expected a string argument was given a numeric one or vice versa.

Error Message	What caused the error and how to fix it
TYPE MISMATCH (cont'd)	Can't mix statement types, so change one side of the assignment statement so it agrees with the other side (sides meet at the = sign). Check the function argument types and use the correct type (numeric or string).
UNDEFINED STATEMENT	An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist. Insert the necessary statement number or branch to another statement number.
UNDEFINED USER FUNCTION	Reference was made to a user-defined function which had never been defined. Define the function.
FILE OPEN	You have attempted to open a previously opened file. Check logical file numbers (1st parameter in the OPEN statement) and be sure you use unique numbers for each file.

Error Message	What caused the error and how to fix it
FILE NOT OPEN	You have attempted to read from, write to, or close a file not previously opened. Open the file.
NOT INPUT FILE	You tried to INPUT# from a file opened for writing. Reading requires a Ø as the 3rd parameter of the OPEN statement. Read (Ø) is the default option.
NOT OUTPUT FILE	You tried to PRINT# to a file opened for reading. Writing to a file requires a 1 (or a 2 if you want an EOT at the end of the file) as the 3rd parameter in the OPEN statement.
DEVICE NOT PRESENT	You have attempted to open a file on a device which is 'invisible' to PET. Check device numbers (2nd parameter in the OPEN statement) and be sure the device is assigned and connected properly and turned on.