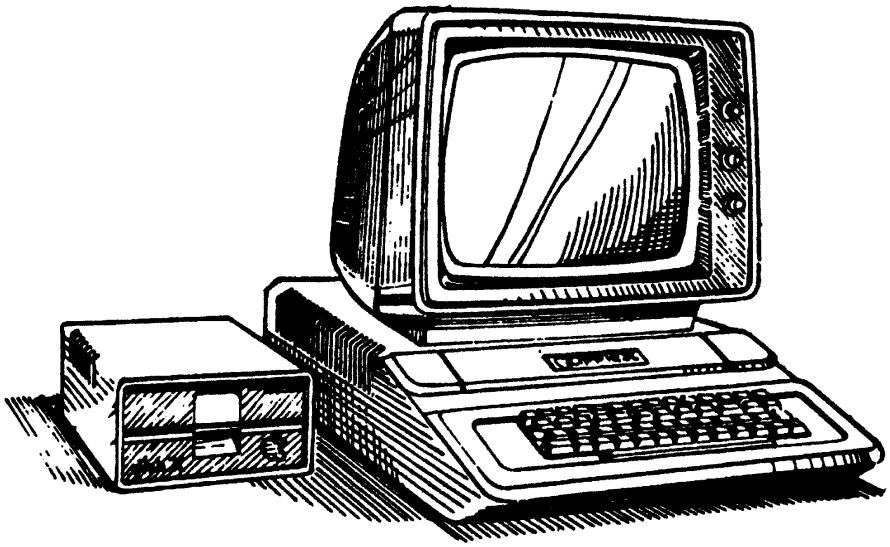


APPLESOFT BASIC
Extended Precision Floating Point
BASIC Language
Reference Manual

[Cassette - RAM Version]

AUTHOR
Apple Computer Inc.

DOCUMENT DATES OF RECORD
November 1977

(This page is not part of the original document)

INTRODUCTION

What is this manual?

The APPLESOFT BASIC Extended Precision Floating Point BASIC Language Reference Manual describes Apple Computer, Inc.'s version of the BASIC language for use by its Apple II computer series.

This manual contains detailed information about this computer language for use by Apple II computer programmers.

Facts about this manual

Author:

Apple Computer Inc

Document dates of record:

November 1977

Owner:

Organization:	DigiBarn Computer Museum	(www.digibarn.com)
Curator:	Bruce Damer	(http://www.damer.com/)

(This page is not part of the original document)

PROPERTY STATEMENT

(This page is not part of the original document)

Disclaimer

The Commons Deed is not a license. It is simply a handy reference for understanding the Legal Code (the full license) — it is a human-readable expression of some of its key terms. Think of it as the user-friendly interface to the Legal Code beneath. This Deed itself has no legal value, and its contents do not appear in the actual license.

Creative Commons is not a law firm and does not provide legal services. Distributing of, displaying of, or linking to this Commons Deed does not create an attorney-client relationship.

Cassette - RAM version

APPLESOFT

EXTENDED PRECISION FLOATING POINT BASIC LANGUAGE

REFERENCE MANUAL

NOVEMBER, 1977

APPLESOFT

Table of Contents

APPLESOFT

Table of Contents

Continued

Introduction

APPLESOFT is a powerful, floating point BASIC written expressly for the Apple II computer by Microsoft Inc.

This BASIC is intended for use in business, science and educationally oriented applications which require extensive manipulation of decimal numbers.

This manual provides the Apple II user with a complete description of all APPLESOFT commands with examples of how they are used.

It is assumed that the user already has at least a minimal working knowledge of the BASIC language.

Getting Started

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. For further reading suggestions, see Appendix I.

If your Apple II does not have Floating Point BASIC loaded and running, follow the procedures in Appendice A.

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.

Once your TV has displayed a "] " prompt character, you are ready to use APPLESOFT/BASIC.

NOTE: All commands to APPLESOFT BASIC should end with a carriage return (depressing the "RETURN" key). The carriage return tells BASIC that you have finished typing the command. If you make a typing error, type a back arrow (←). Repeated use of " ← " will eliminate previous characters. Typing a "CTRL"-X will eliminate the entire line that you are typing. See Appendix B for more details on editing.

Direct Commands

Now, try typing in the following:

```
PRINT 10-4  (end with carriage return)
```

Apple II will immediately print:

```
6
```

The print statement you typed in was executed as soon as you hit the carriage return key. BASIC evaluated the formula after the "PRINT" and then typed out its value, in this case 6.

Now try typing in this:

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

BASIC will print:

```
.5          30
```

As you can see, BASIC can do division and multiplication as well as subtraction but did not get 1 divided by 2 exactly right. We will cover calculation errors and how to set number of decimal places later on. 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 3 columns, each 14 characters wide. The last two of the positions on the line are not used. The result is a " , " causes BASIC to skip to the next 14 column field on the terminal, where the value 30 was printed.

Indirect Commands

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 an integer from 0 to 65529.

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, APPLE/SOFT BASIC saves Indirect Commands in the Apple's memory. When you type in "RUN", BASIC will execute the lowest numbered indirect statement first, then the next highest, etc. for as many as were typed in.

Suppose we type in "RUN" now (remember to depress "RETURN" key at the end of each line you type):

```
RUN
```

Apple will now display on your TV:

```
 5
-1
```

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
```

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
```

Apple will reply with:

```
20 PRINT 2-3
```

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
```

Apple will reply with

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

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
```

Apple will reply with:

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

Number Format

We will digress for a moment to explain the format of numbers in APPLESOFT 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 APPLESOFT BASIC. is $1.Ø*1Ø^{38}$, while the smallest positive number is $1.Ø*1Ø^{-39}$.

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 Ø 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 an integer Ø to 9)

The leading "S" is the sign of the number, a space for a positive number and a " - " for a negative one. One nonzero 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 1Ø 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. BASIC checks to see if the entire number will fit on the current line. If not, a carriage return/line feed is executed before printing the number.

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 type 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
```

Apple will reply with:

```
]
```

Now type in:

```
LIST
```

BASIC will reply with:

```
]
```

Color Graphics Example

Now type in:

```
PLTG
```

This will black out the top twenty lines on your TV screen and leave only four lines of text at the bottom. Your Apple is now in its "Color Graphics" mode.

Now type in:

```
PLTC 13
```

Apple Basic will only respond with a "]" and a flashing cursor but internally you have selected a yellow color

Now type in:

```
PLTP 20, 20
```

Apple will respond by plotting a small yellow square in the center of the screen.

Now type in:

```
PLTH 0, 30, 20
```

Apple will draw a horizontal line from the left edge of the screen to one-quarter of a screen width of the right and one-quarter down from the top.

Now type in:

```
PLTC 6
```

To change to a new color and then type in:

```
PLTV 1Ø, 39, 3Ø
```

More about Color Graphics later. To get back to all text mode, type in:

```
TEX
```

The characters on the screen is Apple's attempt to display color information as TEXT.

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          .333333312
```

Print Format

As explained earlier, including a " , " in a print statement causes it to space over to the next fourteen 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 1,2,3
     1             2             3

B)  PRINT 1;2;3
     1  2  3

C)  PRINT -1;2;-3
    -1  2  -3
```

The following are examples of various numbers and the output format Apple will use to print them:

NUMBER	OUTPUT FORMAT
+1	1
-1	-1
6523	6523
-23.46Ø	-23.46
1x1Ø 2Ø	1E+2Ø
-12.34567896x1Ø1Ø	-1.23456787E-Ø6
1.23456789E-7	1.2345678E-1Ø
1ØØØØØØØØØ	1E+Ø9
999999999	999999999
.1	.Ø99999992

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 (255 characters). However, only the first 1Ø digits are significant, and the tenth digit is rounded up.

```
PRINT 1.23456784912345678

1.23456785
```

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:

```
1Ø INPUT R
2Ø PRINT 3.14159*R*R
RUN
? 1Ø
 314.159ØØ2
```

Here's what's happening. When BASIC encounters the "INPUT" statement, it outputs a question mark (?) and then waits for you to type in a number. When you do (in the above example 1Ø was typed), execution continues with the next statement in the program after the variable (R) has been set (in this case to 1Ø). In the above example, line 2Ø would now be executed. When the formula after the PRINT statement is evaluated, the value 1Ø is substituted for the variable R each time R appears in the formula. Therefore, the formula becomes 3.14159*1Ø*1Ø, or 314.159.

If you haven't 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's an easier way to do it simply by adding another (line 3Ø) to the program as follows:

```
3Ø GOTO 1Ø
RUN
? 1Ø
 314.159ØØ2
```

```
? 3
 28.27431
? 4.7
 69.3977229
?
```

By putting a "GOTO" statement on the end of our program, we have caused it to go back to line 1Ø 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 carriage return to the input statement (thus a blank line).

Variable Names

The letter "R" in the program we just ran 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. An alphanumeric character is any letter (A-Z) or any number (Ø-9).

Below are some examples of legal and illegal variable names:

LEGAL	ILLEGAL
TP	TO (variable names cannot be reserved words)
PSTG$	
COUNT	RGOTO (variable names cannot contain reserved words)
N1%	

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

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

ABS AND ASC ATN CHR$ CLEAR CONT COS DATA DEF

DIM END EXP FN FOR FRE GOSUB GOTO IF IN

INPUT INT LEFT$ LEN LET LIST LOAD LOG MID$ NEW

NEXT NOT ON OR OUT PEEK PLT PLTC PLTG PLTH PLTP

PLTV POKE POS PRINT READ REM RESTORE RETURN

RIGHT$ RND RUN SAVE SGN SIN SPC SQR STEP STOP

STR$ TAB TAN TEX THEN TO USR VAL

Assigning Variable Values

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.

Try the following examples:

```
A=5

PRINT A,A*2
 5            1Ø

LET Z=7   (can be used only if Option 2 was loaded.)

PRINT Z, Z-A
 7            2
```

As can be seen from the examples, the "LET" is optional in an assignment statement only if option 2 is loaded initially.

BASIC "remembers" the values that have been assigned to variables using this type of statement. This "remembering" process uses space in the Apple II's 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 CLEAR 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 formula. Try the example below:

```
PRINT Q,Q+2,Q*2
 Ø            2          Ø
```

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. It is not available in Option 1.

IF...THEN

Suppose we wanted 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 Apple II and run, it will ask for a value for B. Type in any value you wish. The Apple 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 skip to 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 (remember to type "NEW" first to delete your last program):

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

When this program is run, line 10 will ask for two numbers to be entered from the keyboard. 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 LARGER" and then line 40 sends the computer back to line 10 to begin again.

At line 2Ø, if A has the same value as B, A<=B is true so we go to line 5Ø. At line 5Ø, 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 7Ø sends us back to the beginning again.

At line 2Ø, if A is smaller than B, A<=B is true so we go to line 5Ø. At line 5Ø, A<B will be true so we then go to line 8Ø. "B IS LARGER" 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 carriage return to the input statement.

Another Color Example

Let's try another program. The one below uses another form of "If...THEN"; i.e. "IF" statement 1 is true "THEN" let statement 2 be executed otherwise go the next line number. After you type in the program below, "LIST" it and make sure that you have typed it in correctly. Now "RUN" it.

```
1Ø   PLTG
2Ø   NX=Ø:NY=Ø:X=Ø:Y=Ø:XV=2:YV=1
3Ø   T9=39:TØ=Ø:COLR=13:J=1:K=25Ø
4Ø   NX=X+XV:NY=Y+YV
5Ø   IFNX>=T9 THEN NX=T9
6Ø   IFNX<=TØ THEN NX=TØ
7Ø   IFNY>=T9 THEN NY=T9
8Ø   IFNY<=TØ THEN NY=TØ
9Ø   IFNX=T9 or NX=TØ THEN XV=XV
1ØØ  IFNY=T9 or NY-TØ THEN YV=YV
11Ø  PLTC=COLR: PLTP NX,NY
12Ø  PLTC=TØ: PLTP X,Y
13Ø  X=NX: Y=NY
14Ø  I=I+J:IFI<K THEN 4Ø
15Ø  TEX
16Ø  PRINT "FINISHED"
```

As you have seen, Apple can do more than just use numbers. We'll return to color graphics again after you have learned more about APPLESOFT BASIC.

FOR....NEXT

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 1Ø. 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:

```
1Ø  PRINT 1,SQR(1)
2Ø  PRINT 2,SQR(2)
3Ø  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 the result of 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 in 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" 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 the "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 12 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 N
```

Notice that the major difference between this program and the previous one using "FOR" loops is the addition of the STEP
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-1
40  IF I>=1 THEN 20
```

Notice that we are now checking to see that I is greater than 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 less than 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 J
50  NEXT I
```

Notice that the "NEXT J" comes before the "NEXT I". This is because the J-loop is inside of the I-loop. The following program is incorrect; run it and see what happens.

```
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.

Matrices

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 Ø to 15. Matrix subscripts always start at Ø; 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 (Ø through 1Ø).

As an example of how matrices are used, try the following program 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=Ø
 80  FOR I=1 TO 7
 90  IF A(I)<=A(I+1) THEN 14Ø
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.

GOSUB....RETURN

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 the "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 error would occur. Each "GOSUB" executed in a program should have a matching "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, "END" will return to command mode as indicated by a "] " and a flashing cursor.

READ...DATA ...RESTORE

Suppose you had to enter numbers to your program that didn't 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,-39,28,391,-8,0,3.14,90
120  DATA 89,5,10,15,-34,-999999
```

This is what 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 of 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 statement, an "OUT OF DATA" 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 of 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's 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.

Real, Integer, and String Variables

There are three different values used in APPLESOFT BASIC. So far we have just used one type - real precision. Numbers in this mode are displayed with up to nine decimal digits of accuracy and may range up to 10 to the 38th power. Apple converts your numbers from decimal to binary for its internal use and then back to decimal when you ask it to "PRINT" the answer. Internal math routines such as square root, divide, exponent do not always give the exact number that you expected. For example:

```
PRINT SQR(4)    gives 1.99999982 not 2
PRINT 1/2       gives .500000032 not .5
PRINT 10^3      gives 999.998962 not 1000
```

The number of places to the right of the decimal point may be set by rounding off the value prior to printing it. The general formula is:

```
X=  INT(X*10 D+.5)/INT(10^D+.5)
```

Where D is the number of decimal places, a faster way to set the number of decimal places is to use the formula:

```
X=  INT(X*D+.5)/D
```

Where D=10 is one place; D=100, 2 places; D=1000, 3 places, etc. The above works for X>=1 and X<999999999. A routine to limit the number of digits after the decimal point is given in the section on string functions.

The table below summarizes the three types of values used in APPLESOFT BASIC programming:

DESCRIPTION	SYMBOL to Append to Variable Name	EXAMPLE
Strings (Ø to 255 characters)	$	A$ ALPHA$
Integers (must be in range of -32767 to +32767)	%	B% C1%
Real Precision (exponent:-38 to +38, with 9 decimal digits)	none	C BOY

An integer or string variable must be followed by a "%" or "$" at each use of that variable. For example X, X%, and X$ are each different variables.

Integer variables are not allowed in "FOR" or "DEF" statements. The greatest advantage of integer variables is their use in matrix operations wherever possible to save storage space and provide fastest execution.

All arithmetic operations are done in floating point. No matter what the operands to +,-,*,/, and ↑ are, they will be converted to floating point. The functions SIN, COS, ATN, TAN, SQR, LOG, EXP and RND also convert their arguments to floating point and give the result as such.

The operators AND, OR, NOT force both operands to be integers between -32767 and +32767 before the operation occurs.

When a number is converted to an integer, it is truncated (rounded down). For example:

```
I%=.999              A%= -.01
PRINT I%             PRINT A%
Ø                    -1
```

It will perform as if INT function was applied. No automatic conversion is done between strings and numbers.

Strings

A list of characters is referred to as a "String". BILL, APPLE, and THIS IS A TEST are all strings. Like numberic 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$= "GOOD MORNING"

PRINT A$
GOOD MORNING
```

In this example, we set the string variable A$ to the string value "GOOD MORNING". 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("YES")
 12           3
```

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 Ø 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 terminal 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
```

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 part of a string and manipulate it. Now that we have set A$ to "GOOD MORNING", we might want to print out only the first four characters of A$. We would do so like this:

```
PRINT LEFT$(A$,4)
GOOD
```

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

```
FOR N=1 TO LEN(A$):PRINT LEFT$(A$,N):NEXT N
G
GO
GOO
GOOD
GOOD
GOOD M
GOOD MO
GOOD MOR
GOOD MORN
GOOD MORNI
GOOD MORNIN
GOOD MORNING
```

Since A$ has 12 characters, this loop will be executed with N=1,2, 3...,11,12. 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
```

"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

G               GO
O               OO
O               OD
D               D
                 M
M               MO
O               OR
R               RN
N               NI
I               IG
G
```

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$=A$+" "+"BILL"

PRINT B$
GOOD MORNING BILL
```

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

```
C$=RIGHT$(B$,3)+"-"+LEFT$(B$,4)+"-"+MID$(B$,6,7)

PRINT C$
BILL-GOOD-MORNING
```

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"

PRINT VAL(STRING$)
 567.8
```

```
STRING$=STR$(3.1415)

PRINT STRING$, LEFT$(STRING$,5)
 3.1415         3.14
```

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

The following short program demonstrates how string functions may be used to format output of numeric variables:

```
100 INPUT "ENTER ANY NUMBER",X
110 INPUT "ENTER NO. OF DIGITS TO RIGHT OF
      DECIMAL PT.";D
120 GOSUB 1000
130 PRINT "***"
140 GO TO 100
1000 X$=STR$(X):FOR I = 1 TO LEN (X$)^+1:
     IF MID$ (X$,I,1,) < > "E" THEN NEXT
1010 FOR J=1 TO I-1: IF MID$ (X$,J,1)< > "."
     THEN NEXT
1020 PRINT LEFT $ (X$, -(J+D)*(J+D<=I-1)-
     (I-1)*(J+D>I-1))+MID$(X$,I);:RETURN
```

The above program uses a subroutine starting at line 1000 to print out a predefined variable X with D digits after the decimal point. Answer is truncated; not rounded off. The variables X%, I and J are used in the subroutine as local variables. Line 1000 converts variable X to string variable X$ and scans the string to see if an "E" is present. I is set to the position of the "E" or to LEN(X$)+1 if no "E" is there. Line 1010 searches the string for a decimal point and sets J equal to its position. Line 1020 prints out variable X as a string with no trailing spaces and no carriage return. The "LEFT$" function prints out significant digits and the "MID$" function prints out exponent if it was there. The relational expressions inside the "LEFT$" check to see if at least D digits to the right of the decimal point are available to be printed.

"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
```

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 H

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$(15)
110 FOR I=1 TO 15:READ A$(I):NEXT I:
120 F=0:I=1
130 IF A$(I) =A$(I+1) THEN 180
140 T$=A$(I+1)
150 A$(I+1)=A$(I)
160 A$(I)=T$
170 F=1
180 I+1: IF I 15 GOTO 130
190 IF F THEN 120
200 FOR I=1 TO 15:PRINT A$(I):NEXT I
220 DATA APPLE,DOG,CAT,RANDOM,COMPUTER,BASIC
230 DATA MONDAY,"***ANSWER***","FOO: "
240 DATA COMPUTER,  FOO,ELP,MILWAUKEE,SEATTLE,ALBUQUERQUE
```

Color Graphics

In two previous examples on pages 5 and 11, Apple II has demonstrated its ability to do color graphics as well as text. In color graphics mode, Apple displays an array of 16ØØ small squares in 16 colors on a 4Ø by 4Ø grid plus provides 4 lines of text at the bottom of the screen. The horizontal or X axis is standard with Ø the left most position and 39, the right most. The vertical or Y axis is non-standard in that it is inverted; i.e., Ø is the top most position and 39, the bottom most.

Warning: If Apple is instructed to plot color graphics outside of the Ø to 39 range, it will do so. This may clobber your program or the APPLESOFT complier.

Type in the following demonstration program; remember to type "NEW" first.

```
  10 PLTG
  20 Z=USR(-936)
  30 PRINT"INITIALIZE COLOR GRAPHICS; SE
T40X40 TO"
  40 PRINT"BLACK. SET WINDOW TO 4 LINES
AT BOTTOM"
  50 PRINT"CLEAR ALL TEXT"
  60 GOSUB1000
  70 PLTC 2:PLTP 0,0
  80 Z=USR(-936):PRINT"COLOR=BLUE; PLOT
AT 0,0"
  90 GOSUB1000
  100 PLTC  1:PLTP 39,0
  110 Z=USR(-936):PRINT"COLOR=MAGENTA; P
LOT AT 39,0"
  120 GOSUB1000
  130 PLTC 12:PLTP 0,39
  140 Z=USR(-936):PRINT"COLOR=GREEN; PLO
T AT 0,39"
  150 GOSUB1000
  160 PLTC 9:PLTP39,39
  170 Z=USR(-936):PRINT"COLOR=ORANGE; PL
OT AT 39,39"
  180 GOSUB1000
  190 PLTC 12:PLTP19,19
  200 Z=USR(-936):PRINT"COLOR=YELLOW; PL
OT AT 19,19"
  210 GOSUB1000
```

```
  220 Z=USR(-936)
  230 PRINT"PLOT YOUR OWN POINTS"
  240 PRINT"REMEMBER X&Y MUST BE >=0 & <
=39"
  250 INPUT"ENTER X,Y";X,Y
  260 IFX>39 OR X<0 THEN 230
  270 IF Y<0 OR Y>39 THEN 230
  280 PLTC 8:PLTP X,Y
  290 PRINT"HIT RETURN TO STOP":GOTO250
  1000 PRINT"***HIT ANY KEY TO CONTINUE*
**";:GETA$:RETURN
```

After you have typed it in, "LIST" it and check for typing errors. You may want to "SAVE" it on cassette tape for future use. Now "RUN" the program.

The program uses four new commands:

```
PLTG
PLTC I
PLTP X, Y
Z = USR (-936)
```

The command "PLTG" tells Apple to switch to its color graphics mode. It also clears the 40 by 40 plotting area to black, sets the text output to be limited to a window at the bottom of the screen of 4 lines of 40 characters each and sets next color to be plotted to black.

"PLTC I" command sets the next color to be plotted to the value of expression I. Color remains set until changed by a new "PLTC" command. For example, the color plotted in line 280 remains the same no matter how many points are plotted. The value of expression I must be in the range of 0 to 15 or an error may occur.

Change the program by re-typing in lines 250 and 280 as follows:

```
250 INPUT "ENTER X, Y, COLOR"; X, Y, Z
280 PLTC C: PLTP X, Y
```

Now "RUN" the program and you will be able to select your own colors as well as points. We will demonstrate Apple's color range in a moment.

"PLTP X, Y" command plots a small square of color defined by the last "PLTC" command at the position specified by expressions X and Y. Remember, X and Y must each evaluate to a number in the range of 0 to 39.

"Z = USR(-936)" is a useful function used to clear the text area and set the cursor to the top left of the currently defined text window so that the next text output will start at that position. In color graphics mode, this would be the beginning of line 20 since lines Ø through 19 are now being used for color graphics plotting area.

Note: To get from color graphics back to all text mode, type "TEX" and depress "RETURN" key if you have the "⌐" prompt character.

Type in the following program and "RUN" it to display Apple's range of colors ("NEW" first).

```
 10 PLTG:Z=USR(-936)
 20 FOR I=0TO31
 30 PLTC I/2
 40 PLTV0,39,I
 50 NEXT
 60 FORI=0TO8 STEP2:PRINTTAB(I*2);I;:NE
XT
 70 FORI=10TO14STEP2:PRINTTAB(I*2-1);I;
:NEXT
 80 PRINT:FORI=1TO9STEP2:PRINTTAB(I*2);
I;:NEXT
 90 FORI=11TO15STEP2:PRINTTAB(I*2-1);I;
:NEXT:PRINT
 100 PRINT"STANDARD APPLE COLOR BARS";
```

Color bars are displayed at double their normal width. The left most bar is black as set by PLTC Ø; the right most, white, is set by PLTC 15. Depending on the tint setting on your TV, the second bar as set by PLTC1 will be magenta (reddish-purple) and the third will be blue. Adjust your TV tint control for these colors.

In the last program a new command "PLTV Y1, Y2, X" was used. This command plots a vertical line from the Y coordinate specified by expression Y1 to expression Y2 at the horizontal position specified by expression X. Y1, Y2 and X must evaluate to values in the range of Ø to 39. In addition Y2 must be greater than or equal to Y1. The command "PLTH X1, X2, Y " is similar to "PLTV" except that it plots a horizontal line.

Note: Apple draws an entire line just as fast as it plots a single point!

The next example program, although long, illustrates Apple's ability to animate in color graphics mode. After typing the program in, "LIST" it by sections to check for typing errors. Now "SAVE" it on cassette tape before "RUN"ing it. Do this because if you made a typo error that you did not catch, "RUN"ing the program may clobber it.

```
  10 GOTO 1000
  20 IF PEEK(C7)<128 THEN RETURN
  30 POKE-16368,0:GOTO 1070
  100 PLTC CL%(I):D=C6:IF RND(X+Y+C9)*N<
C9THENI=ABS(FNMD(I+RND(X+C8)*D3-C3))
  110 PLTP X,Y:D=C2
  120 X=ABS(FNMD(X+SGN(RND(Y+C8)-D2)*(RN
D(X+C9)+C3+D2)))
  130 D=D6:Y=ABS(FNMD(Y+SGN(RND(X+C8)-D2
)*(RND(Y+C9)+C3+D2)))
  140 GOSUB 20:GOTO 100
  200 PLTG:D=USR(-936)
  205 FOR X=0 TO 38
  210 D=C2:I=FNMD(I+C3):J=I
  220 D=C6:PLTC CL%(FNMD(I+N))
  230 PLTH C1,C4,J:PLTV C1,C4,C4-I
  240 PLTC CL%(FNMD(J+N))
  250 PLTV C1,C4,J:PLTH C1,C4,C4-I
  260 NEXT:N=N+1
  270 GOSUB 20:GOTO 205
  300 PLTG:D=USR(-936)
  305 D=D+1:IF D>10 THEN D=0
  310 FOR I=1 TO 5:FOR J=1 TO 4
  320 PLTC J+D:N=J*C8+D1+I:X=C4-N
  330 PLTH X,N,X:PLTV X,N,N:PLTH X,N,N:P
LTV X,N,X
  340 NEXT:NEXT
  350 GOSUB 20:GOTO 305
```

(Program continued on next page.)

```
  1000 I=0:J=0:N=0:X=0:Y=0:D=0
  1010 C0=128:C1=0:C2=40:C3=1:C4=39:C5=3
8:C6=15:C7=-16384:C8=5:C9=4
  1020 D1=14:D2=.43:D3=3.1:D4=150:D5=10
:D6=48
  1030 DEF FNMD(X)=X-D*INT(X/D)
  1040 DATA 0,8,9,13,15,11,3,2,6,7,14,12
,4,5,0,0
  1050 DIM CL%(15)
  1060 FOR D= 0 TO 15:READ X:CL%(D)=X:NE
XT
  1070 TEX
  1075 D=USR(-936)
  1080 PRINT:PRINT
  1090 PRINT"COLOR DEMOS":PRINT
  1100 PRINT"1 HARMONIC COLOR BARS"
  1110 PRINT"2 CROSSES"
  1120 PRINT"3 TUNNEL"
  1130 PRINT"4 SPLOTCH"
  1135 PRINT"5 EXIT DEMOS":PRINT
  1140 PRINT"HIT ANY KEY TO RETURN TO TH
IS MENU WHILE DEMO IS RUNNING":PRINT
  1150 PRINT"ENTER DEMO NUMBER=";:GETA$
  1160 D=VAL(A$):ON D GOTO 3100,200,300,
4000,2000
  1170 PRINTA$:PRINT"USE ONLY NOS. 1 THR
U 5":GOTO1080
  2000 TEX
  2010 D=USR(-936)
  2020 STOP
  3100 PLTG:D=USR(-936)
  3110 FOR I= 0 TO 31
  3120 PLTC I/2:PLTV 0,19,I
  3130 PLTC CL%(I/2):PLTV 20,39,I
  3140 NEXT
  3150 PRINT"STANDARD COLORS ON TOP"
  3160 PRINT"HARMONIC COLORS ON BOTTOM"
  3170 GET A$:GOTO 1070
```

(Program continuted on next page.)

```
 4000 X=19:Y=23:I=RND(X)*14+1
 4010 D=USR(-936):PRINT:PRINT
 4020 PRINT"BACKGROUND COLOR - 1 THRU 9
?";:GET A$:PRINTA$
 4030 D=VAL(A$):IF D=0 THEN 4020
 4040 PRINT"SPLOTCH SIZE - 1 THRU 9 ?";
:GETA$:PRINT A$:N=VAL(A$)*50
 4050 IF N<1 THEN 4040
 4060 PLTG:POKE-16302,0
 4070 PLTC D:FOR D=47 TO 0 STEP -1:PLTH
0,39,D:NEXT
 4080 GOTO 100
```

The program is not written in the order that it executes. This is to establish maximum operating speed as is outlined in Appendix E. A frequently used subroutine is located as early in the program as possible at lines 2Ø and 3Ø. Three demo programs are next in lines 1ØØ - 14Ø, 2ØØ - 27Ø, and 3ØØ - 35Ø.

Line 1ØØØ defines most frequently used variables, then frequently used constants are defined as variables. Lines 17Ø - 117Ø are the introduction and instructions. A non-animated demo program is located last in lines 31ØØ - 317Ø.

Now run the program and select demo #1. Harmonic colors is just a different sequence in displaying standard colors. Adjacent colors are color coordinated. This different sequence is established in the number array defined in lines 1Ø4Ø - 1Ø6Ø. We will use this in the other examples.

To keep variables within the maximum allowable range, we define a user program in line 1Ø3Ø:

```
1Ø3Ø DEF FNMD(X) = X-D*INT(X/D)
```

This function calculates X modulo D where D is a pre-defined variable.

"CROSSES" demo is located at lines 2ØØ - 27Ø and plots a sequence of horizontal and vertical lines.

"TUNNEL" demo is located at lines 3ØØ - 35Ø and plots a sequence of squares.

"SPLOTCH" demo is located at lines 1ØØ - 14Ø and is initialized by lines 4ØØØ - 4Ø8Ø. It plots a point; then moves at random one or two positions in any direction then plots again. "SPLOTCH SIZE" input is used to establish a probability that the color of the next point will be changed.

Try out each of the demos.

In "SPLOTCH", Apple has illustrated another color graphics feature. The four lines of text at the bottom of the screen have been eliminated and the color field is now 40 by 48 in size. The vertical axis Y can now range from 0 to 47. This switch was done by the POKE in line 4060:

```
4060 PLTG: POKE - 16302,0
```

Note: From all color graphics (40 x 48) display mode, "PLTG" command will switch Apple to mixed color graphics mode (40 x 40 plus 4 lines of text) and "TEX" command will switch to all text mode.

The next example illustrates how Apple can read back the color of a given screen location. This is the reverse of a "PLTP" command. This routine will, given an X, Y position, tell the computer what color was plotted at that location sometime in the past. Type in and "RUN" the following program.

```
  10 GOTO 500
  20 NX=X+XV:NY=Y+YV/C1
  30 POKE D3,NX:POKE D2,NY:J=USR(D1)
  40 IF PEEK(D4)=C2 GOTO 80
  50 PLTC C1:PLTP NX,NY
  60 PLTC C0:PLTP X,Y
  70 X=NX:Y=NY:GOTO 20
  80 XV=-XV:YV=-YV
  90 POKE D3,X:J=USR(D1):IF PEEK(D4)=C2
THEN XV=-XV
 100 POKE D3,NX:POKE D2,Y:J=USR(D1):IF
PEEK(D4)=C2 THEN YV=-YV
 110 IF PEEK(D6)<D5 GOTO 20
 120 POKE-16368,0:TEX
 130 J=USR(-936):STOP
```

(program continuted on next page.)

```
  500 NX=0:NY=0:X=3:Y=2:XV=1:YV=2:CL=2:J
=0
  510 C0=0:C1=3:CL=15:C2=2
  520 D1=784:D2=785:D3=787:D4=810:D5=127
:D6=-16384
  530 DATA169,0,160,0,32,113,248,141,42,
3,96
  540 FOR I= 784 TO 794:READ J:POKE I,J:
NEXT
  550 PLTG:J=USR(-936):PLTC C2
  560 PLTH 0,39,0:PLTV 0,39,39:PLTV 0,39
,0:PLTH 0,39,39
  570 FOR I= 0 TO 3
  580 PLTH 8,13,I+6:PLTV 26,32,I+26:NEXT

  590 FOR I= 0 TO 5
  600 PLTV 20,35,I+9:PLTH 24,34,I+12:NEX
T
  610 PRINT"HIT ANY KEY TO STOP"
  620 GOTO 20
```

The main animating program is located in lines 20 - 110. Compare this with the IF...THEN ricochet program you tried on page 11.

Lines 500 - 620 do housekeeping and are used only once (See Speed Hints Appendix). Line 500 defines most frequently .variables. Lines 510 and 520 define frequently used constants as variables. Data statement in line 530 contains the decimal values for a machine language subroutine which is used to read the screen color. Line 540 "POKE" 's this program into memory starting at hexidecimal location $310. The following routine is then used in APPLESOFT BASIC to read screen color:

```
POKE 787,X     SETS X
POKE 785,Y     SETS Y
J=USR(784)     CALL SUBROUTINE
I=PEEK(810)    FETCH COLOR OF X,Y
```

The advantage of this routine is that the BASIC program does not have to keep track of where every object is on the screen.

Here is how the program works:

Line #		DESCRIPTION
1Ø	---	Go do housework at 5ØØ
5ØØ-52Ø	---	Define variables for speed
53Ø-54Ø	---	Set up machine language subroutine
55Ø-	---	Initialize graphics mode
56Ø	---	Draw a border around screen
57Ø-6ØØ	---	Draw a square
61Ø	---	Message
62Ø	---	Go to main program
2Ø	---	Begin main program loop. Calculate proposed new ball X and Y coordinates by adding their respective incremental velocities to their old positions.
3Ø	---	Set up to read screen color at proposed new ball position.
4Ø	---	The example on page 11 decided on ball reflection by testing to see if new ball position was out of bounds (39 or Ø) In this program, Apple checks to see if proposed new ball position is the color of an obstacle. If not, program continues to next line. If an object, the program branches to line 8Ø. Variable C2 is the color of the object(s).
5Ø	---	Set color to ball color (CL); plot new ball
6Ø	---	Set color to background color (CØ); erase old ball
7Ø	---	Update coordinates; go back to 2Ø to calculate next move.
8Ø	---	Ball will be on object if plotted on proposed new position, so don't plot it on top of object. Reverse both X and Y velocities so that ball will reverse both X and Y directions. This is valid at a corner but not at a wall.
9Ø	---	Test for non-vertical wall by reading screen at old X but new Y. If true, do not reverse X velocity. (Line 8Ø reversed XV and line 9Ø will reverse it again so that it has sign of original direction.)

Line #		DESCRIPTION
100	---	Same as 90, but test for non-horizontal wall with old X and new Y.
110	---	If key has not been depressed, go back to 20 and calculate next move for ball.
120	---	Key was depressed, so clear keyboard and go to text mode.
130	---	Clear screen and stop

Note: No statements may follow "TEX command on the same line number (see 120 above).

Another built-in feature of Apple is its ability to read the position or value of up to 4 potentiometers (game paddles). The following illustrates how to implement this capability in APPLESOFT. Like the screen color routine APPLESOFT needs a short machine language program which is "POKE'ed in by lines 530 - 540 to read a paddle.

The following format is needed to read a paddle:

POKE 796,N	Where N is paddle number (0-3)
Z=USR(795)	Call Subroutine
X=PEEK(810)	X is now paddle value

Reading a paddle is illustrated in the following program.

```
10 GOTO 500
20 POKED1,C0:J=USR(D0):NX=PEEK(D2)
30 NX=ABS((NX-X0-C3)*C2/X1)
40 IF NX>C2 THEN NX=C2
50 POKED1,C1:J=USR(D0):NY=PEEK(D2)
60 NY=ABS((NY-Y0-C3)*C2/Y1)
70 IF NY>C2 THEN NY=C2
80 PLTC CL:PLTP NX,NY
90 IF NX<>X THEN 130
100 IF NY<>Y THEN 130
110 IF PEEK(D3)>D4 THEN 150
120 GOTO 20
130 PLTC C0:PLTP X,Y
140 X=NX:Y=NY:GOTO 20
150 POKE-16368,0:TEX
160 J=USR(-936):STOP
```

listing continued on next page at statement 500

```
  500 NX=0:NY=0:X=0:Y=0:X0=255:X1=0:Y0=2
55:Y1=0:J=0
  510 C0=0:C1=1:C2=39:CL=8:C3=2
  520 D0=795:D1=796:D2=811:D3= -16384:D4
=127
  530 DATA162,0,32,30,251,152,141,43,3,9
6
  540 FORI=795 TO 804:READ J:POKE I,J:NE
XT
  550 J=USR(-936):PRINT:PRINT:PRINT
  560 PRINT"APPLE WILL NOW CALIBRATE PAD
DLES #0"
  570 PRINT"AND #1. MOVE THEM BACK AND F
ORTH OVER"
  580 PRINT"HIT ANY KEY WHEN THROUGH"
  590 FOR I= 0 TO 500
  600 POKE D1,C0:J=USR(D0):NX=PEEK(D2)
  610 IF NX<X0 THEN X0=NX
  620 IF NX>X1 THEN X1=NX
  630 POKE D1,C1:J=USR(D0):NY=PEEK(D2)
  640 IF NY<Y0 THEN Y0=NY
  650 IF NY>Y1 THEN Y1=NY
  660 IF PEEK(D3)<D4 THEN 590
  665 POKE -16368,0
  670 PRINT:PRINT"PDLS HAVE A RANGE OF A
 PPROX. 0 TO 255"
  675 PRINT
  680 PRINT"YOUR PDL #0 ACTUAL RANGE WAS
 ";X0;" -";X1
  690 PRINT"YOUR PDL #1 ACTUAL RANGE WAS
 ";Y0;" -";Y1
  700 PRINT:PRINT"HIT ANY KEY TO CONTINU
 E":GETA$
  710 PLTG:J=USR(-936)
  720 PRINT:PRINT"PDL #0 IS BALL X POS.;
 PDL #1 IS Y"
  730 GOTO 20
```

For a variation on this program change line # 130 (see Appendix B on Editing) to read:

```
130 PLTC CX : PLTP X, Y
```

Add line 515 as follows:

```
515 CX = 2 : CL = 13
```

Now "RUN" the program. See the change? Experiment. You'll see how easy and fun it is to modify programs. Also try modifying the simple program on the next page.

Use the rest of this page to record your own changes

LINE#	CHANGE	RESULT

For a final example, Apple will demonstrate how to draw intricate patterns with a very simple program. Type in and "RUN" the following:

```
  10 GOTO 100
  20 FORW=C1TOC6:FORI=C1TOC9:FORJ=C0TOC9

  30 PLTC I*C3+J/W:K=I+J
  40 PLTP I,K:PLTP K,I:PLTP C4-I,C4-K:PL
TP C4-K,C4-I
  50 PLTP K,C4-I:PLTP C4-I,K:PLTP I,C4-K
:PLTP C4-K,I
  60 NEXT:NEXT:NEXT:GOTO20
  100 I=0:K=0:J=0:W=0:C0=0:C1=1:C3=4:C4=
40:C6=16:C9=19
  110 PLTG:W=USR(-936):GOTO20
```

All major commands in APPLESOFT have now been illustrated. The reference section and appendices that follow present additional information.

REFERENCE
MATERIAL

COMMANDS

A command is usually given after BASIC has indicated that it is waiting for a command with a " " prompt character and a flashing cursor. They are executed immediately after the "Return" key is depressed. 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. More than one command may be given on the same line if they are separated by a colon (":").

NAME	EXAMPLE	PURPOSE/USE
CLEAR	CLEAR	Zeroes all Variables and Strings
CONT	CONT	Continues program execution after a control-C is typed 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. Computer will keep executing the series of statements over and over, until you intervene or until power to the computer is cut off. If you suspect your program is in an infinite loop, type in a control-C. The line number of the statement BASIC was executing will be typed out. After BASIC has typed out "Break In.." and "] ", 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 terminate 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 won't be able to continue and will get a "CAN'T CONTINUE" error. It is impossible to continue a direct command. CONT always resumes execution in your program when control-C was typed. If a control-C fails to stop program execution, hit the "Reset" key then type "ØG" and depress the "Return" key. This may recover your program.
LIST	LIST X	Lists line "X" if there is one.
	LIST or LIST-	Lists the entire program. If in process, "LIST" may be interrupted by a control-C. BASIC will complete LISTing the current line and will halt with a "BREAK".
	LIST X-	Lists all lines in a program with a line number equal to or greater than "X".
	LIST -X	Lists all of the lines in a program with a line number less than or equal to "X".
	LIST X-Y	Lists all of the lines within a program from X to Y.
LOAD	LOAD	Loads (reads) an APPLESOFT floating point BASIC program from cassette tape. First beep indicates that Apple has found beginning of program on tape. Second beep and a " " prompt character and a flashing cursor on the TV screen indicate that the program has been successfully loaded without an error. If message indicates that error occurred while loading, re-check cassette settings and cables and try again. Note: Programs saved from integer BASIC (">") may not be run directly in floating point ("] ") and vice versa.

NAME	EXAMPLE	
RUN	RUN	Starts execution of the program currently in memory at the lowest numbered statement. RUN deletes all variables (does a CLEAR and RESTORES DATA. If you have stopped your program and wish to continue execution at some point in the program without clearing variables, use a direct GOTO statement to start execution of your program at the desired line.
	RUN 200	Starts RUN at the specified line number
NEW	NEW	Deletes current program and all variables.
SAVE	SAVE	Saves (stores) the current floating point program onto cassette tape. Current program is left unchanged. Apple does not verify that the recorder was running and in "record" mode or that the tape is good. "]" prompt and cursor will return when "SAVE" is complete.
	SAVE:SAVE	Saves a program twice on tape so that if there is a bad spot on the tape on the first one, the second may be able to be retrieved

ARITHMETIC OPERATORS

SYMBOL	SAMPLE STATEMENT	PURPOSE/USE
=	A=100 LET Z=2.5	Assigns a value to a variable. The LET is optional only if option 2 is loaded.
-	B=-A	Negation. Note that 0-A is subtraction, while -A is negation.
↑ (↑ is a shift-n)	130 PRINT X↑3	Exponentiation (equal to X*X*X in the sample statement). 0↑0=1; 0 to any other power = 0;A↑B with A negative and B not an integer gives an "ILLEGAL QUANTITY" error.
*	140 X=R*(B*D)	Multiplication
/	150 PRINT X/1.3	Division
+	160 Z=R+T+Q	Addition
-	170 J=100-I	Subtraction

LOGICAL AND RELATIONAL OPERATORS		PURPOSE/USE
=	10 IF A=15 THEN 40	Expression Equals Expression
<>	70 IF A<>0 THEN 5	Expression Does Not Equal Expression
>	30 IF B>100 THEN 8	Expression Greater Than Expression
<	160 IF B<2 THEN 10	Expression Less Than Expression

LOGICAL AND RELATIONAL OPERATORS (CONT.)

SYMBOL	SAMPLE STATEMENT	PURPOSE/USE
<=,=<	180 IF 100<=B+C THEN 10	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 -1=0 (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
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
NOT	1	-	0
	0	-	1

LOGICAL AND RELATIONAL OPERATORS (CONT.)

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 binary 10000 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 1111....111 and 8 equals binary 1000, so the result is binary 1000 or 8 decimal.
4 AND 2=0	4 equals binary 100 and 1 equals binary 10, so the result is binary 1 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 OR 10=10	Binary 1010 OR'd with binary 1010 equal- binary 1010, or 10 decimal.
-1 OR -2=-1	Binary 111...111 (-1) OR'd with binary 111...1110 (-2) equals binary 111...111, or -1.
NOT 0=-1	The bit complement of binary 0 to 16 places is sixteen ones (1111....1111) 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 complment of 1 is 1111....1110, which is equal to -(1+1) or -2.

The following is a 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.

RULES FOR EVALUATING EXPRESSIONS:

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.

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)	↑	EXPONENTATION
3)	NEGATION	-X WHERE X MAY BE A FORMULA
4)	* /	MULTIPLICATION AND DIVISION
5)	+ -	ADDITION AND SUBTRACTION
6)	RELATIONAL OPERATORS: (equal precedence for all six)	= EQUAL <> NOT EQUAL < LESS THAN > GREATER THAN <= LESS THAN OR EQUAL TO >= GREATER THAN OR EQUAL TO
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.

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 IF X<>0 THEN...

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 an 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 numeric or string value.

A constant is either a number (3.14) or a string literal ("FOO").

STATEMENTS (cont.)

NAME	EXAMPLE	PURPOSE/USE
DATA	1Ø DATA 1,3,-1E3,.Ø4	Specifies data, read from left to right. Information appears in data statements in the same order as it will be read in the program.
	2Ø DATA " FOO,ZOO"	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 double quotes. It is impossible to have a double quote within string data or a string literal; i.e., (""ANYTHING"") is illegal. Use a single quote mark (') instead.
DEF	1ØØ DEF FNA (V)=V/B+C	The user can define functions like the built-in functions (SQR, SGN, ABS, 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, FNKO, FNR2. User defined functions are stricted to one line. A function may be defined to be any expression, but may only have one argument. In the example B & C are variables that are used in the program. Executing the DEF statement defines the function. User defined functions can be redefined by executing another DEF statement for the same function. User defined string functions are not allowed. "V" is called the dummy variable.
	110 Z=FNA (3)	Execution of this statement following the above would cause Z to be set to 3/B+C, but the value of V would be unchanged.
DIM	113 DIM A(3),B(1Ø)	Allocates space for matrices. All matrix elements are set to zero by the DIM statement.
	114 DIM R3(5,5),D$(2,2,2)	Matrices can have more than one dimension. Up to 255 dimensions are allowed, the size of each must be less than 32767, and is limited by total memory available.
	115 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 have as many subscripts as implied in its first use and whose subscripts may range from Ø to 1Ø (eleven elements).
	117 A(8)=4	If this statement was encountered before a DIM statement for A was found in the program it would be as if a DIM A (1Ø) has been executed previous to the execution of line 117. All subscripts start at zero (0), which means that DIM X (1ØØ) 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	3ØØ FOR V=1 to 9.3 STEP .6	(see NEXT statement) V is set equal to the value of the expression following the equal sign, 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 step is the value of the expression following STEP. When the NEXT statement is encountered, the step is added to the variable.

STATEMENTS (cont.)

NAME	EXAMPLE	PURPOSE/USE
FOR	31Ø 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 <= and 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=1O*N TO 3.4/Q STEP SQR(R)	Note that expressions (formulas) 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.
	32Ø FOR V=9 TO 1 STEP-1	When the statement after the NEXT is executed, the loop variable is not necessarily equal to the final value, but is equal to whatever value caused the FOR....NEXT loop to terminate. The statement between the FOR and its corresponding NEXT in both examples above (31Ø & 32Ø) would be executed 9 times.
GET	45Ø GET A	Fetches a single numeric digit from the keyboard without echoing back to TV screen and without the need for depressing the "RETURN" key.
	46Ø GET A$	Same as above but fetches a single ASCII character from keyboard.
GOTO	5Ø GOTO 1ØØ	Branches to the statement specified.
GOSUB	1Ø GOSUB 91Ø	Branches to the specified statement (91Ø) 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	15 IF X<Ø THEN 5	Branches to specified statement if the relation is True.
	20 IF X<Ø THEN PRINT "X LESS THAN Ø"	Executes all of the statements on the remainder of the line after the THEN if the relation is True.
	25 IF X=5 THEN 5Ø:Z=A	WARNING. The "Z=A" will never be executed because if the relation is true, BASIC will branch to line 5Ø. If the relation is false BASIC will proceed to the line after line 25.
	26 IF X<Ø THEN PRINT "ERROR X NEGATIVE": GOTO 35Ø	In this example, if X is less than Ø, the PRINT statement will be executed and then the GOTO statement will branch to line 35Ø. If the X was Ø or positive, BASIC will proceed to execute the lines after line 26.
INPUT	3 INPUT V,W,W2	Requests data from the keyboard (to be typed in). Each value must be separated from the preceeding value by a comma (,). The last value typed should be followed by a carriage return. A "?" is typed as a prompt character. However, 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 requested, the extra data will be ignored and a warning "EXTRA IGNORED" will be printed when this happens. Strings must be input in the same format as they are specified in DIM statements. If the "RETURN" key is depressed without any data, program execution will be halted.

STATEMENTS (cont.)

NAME	EXAMPLE	PURPOSE/USE
INPUT	5 INPUT "VALUE";V	Optionally types a prompt string ("VALUE") before requesting data from the terminal. Typing CONT after an INPUT command has been interrupted will cause execution to resume at the INPUT statement.
LET	300 LET W=X	Assigns a value to a variable and is optional. (option 2 only)
	310 V=5.1	"LET" is not allowed for option 1 with color graphics.
NEXT	340 NEXT V	Marks the end of a FOR loop.
	345 NEXT	If no variable is given, matches the most recent FOR loop. Executes faster than example in line 340.
	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 the 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=1 or I attempts to select a nonexistent line (>=5) in this case, the statement after the ON statement is executed. However, if I is >255 or <0, an "ILLEGAL QUANTITY" error message will result. As many line numbers as will fit on a line can follow an ON...GOTO
	105 ON SGN (X) +2 GOTO 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 GOSUB branches to the statement after the ON...GOSUB.
PLTC	510 PLTC I	(Option 1 only) Sets TV display color to value in expression I. Expression I must be in the range of 0 to 15. Colors are assigned the values: 0 - Black, 8 - Brown 1 - Magenta, 9 - Orange 2 - Dark Blue, 10 - Grey 3 - Light Green, 11 - Pink 4 - Dark Green, 12 - Green 5 - Grey, 13 - Yellow 6 - Medium Blue, 14 - Blue/Green 7 - Light Blue, 15 - White Color remains set until a new "PLTC" command changes it or until a "PLTG" command clears screen and sets PLTC 0.

STATEMENTS (cont.)

NAME	EXAMPLE	PURPOSE/USE
PLTG	53Ø PLTG	(Option 1 only) Switches TV screen display from all text mode into color graphics (40x40) with 4 lines of textat bottom of screen.
	55Ø PLTG:POKE-16302,Ø	Sets all color graphics (40x48) mode with no text at bottom.
PLTH	57Ø PLTH X1, X2, Y	(Option 1 only) If in color graphics mode, this command draws a horizontal line, of color as set by "PLTC", from coordinate X1 to X2 at position Y. Numeric value for X1, X2 and Y must be between Ø and 39 or program may blow up. (Y may range up to 47 if in all color modes; i.e., no 4 lines of text at bottom of screen.
	59Ø PLTH Ø, 19, Ø	Draws horizontal line along the top of the TV screen from upper-left corner to center of screen.
	61Ø PLTH 2Ø, 39, 39	Draws horizontal line along the bottom of the TV screen from bottom center to lower-right corner.
PLTP	63Ø PLTP X, Y	(Option 1 only) Plots a small square of color set by "PLTC" at coordinates specified by expressions X and Y. Value of X must be between Ø and 39 and Y between Ø and 39 or Ø and 47.
	65Ø PLTP 2Ø, 2Ø	Plots a small square at center of screen.
PLTV	67Ø PLTV Y1, Y2, X	(Option 1 only) Draws a vertical line, of color set by "PLTC" from Y1 to Y2 at X. See "PLTH".
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 =>Ø and <=255, or an "ILLEGAL QUANTITY" error will occur. The address (I) must be =>-65535 and <=65535, or an "ILLEGAL QUANTITY" error will result.
PRINT	36Ø PRINT X,Y,Z, 37Ø PRINT 38Ø PRINT X,Y 39Ø PRINT "VALUE IS";A 4ØØ PRINT A2,B, 41Ø PRINT MID$(A$,2) 42Ø ? XY,Z	Prints the value of expressions on the terminal. If the list of values 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. If a comma appears after an expression in the list, then spaces are outputted until the beginning of the next column field is reached. If there is no list of expression to be printed, then a carriage return is executed. String expressions may be printed. A question mark is the same as a "PRINT" command.
READ	49Ø READ V,W	Reads data into specified variables from a DATA statement. The first piece of data read will be the first peice 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. The line number given in the "SYNTAX ERROR" will refer to the line number where the error actually is located.

STATEMENTS (cont.)

NAME	EXAMPLE	PURPOSES/USE
REM	5ØØ REM NOW SET V=Ø	(Option 2 only) 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 ":".
	51Ø REM SET V=Ø: V=Ø	In this case the V=0 will never be executed by BASIC.
	52Ø V=Ø: REM SET V=Ø	In this case V=0 will be executed.
	53Ø GOTO54Ø: SET V=Ø	(Option 1) This format may be used to simulate a "REM" in programs using graphics commands. Where the "REM" statement is not available.
RESTORE	6ØØ 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 of data read will be the second piece listed in the first DATA statement, and so on as in a normal READ operation.
RETURN	7ØØ RETURN	Causes a subroutine to return to the statement after the most recently executed GOSUB.
STOP	9ØØØ STOP	Causes a program to stop execution and to enter command mode. Prints BREAK IN LINE 9000 (as per this example). CONT after a STOP branches to the statement following the STOP.
TEX	8ØØ TEX	(Option 1 only) Sets TV display to all text mode from color graphics mode and resets TV display to 24 lines of 4Ø characters each if otherwise.
	81Ø TEX: GOTO 5Ø	Illegal....no statements may follow "TEX" on same line number
	82Ø X=USR(-1233)	(Option 1 or 2) Equivalent to "TEX" but works also in Option 2.

INTRINSIC FUNCTIONS

NAME	EXAMPLE	PURPOSE/USE
ABS (X)	12Ø PRINT ABS (X)	Gives the absolute value of the expression X. ABS return X if X>=Ø, -X otherwise.
ATN	13Ø PRINT ATN(X)	Gives the arctangent of the argument X. The result is returned in radians and ranges from $-\pi/2$ to $\pi/2$. ($\pi/2$=1.5708)
COS(X)	14Ø PRINT COS (X)	Gives the cosine of the expression X. X is interpreted as being in radians.
EXP(X)	15Ø PRINT EXP(X)	Gives the constant "E" (2.71828) raised to the power X. (E↑X) The maximum argument that can be passed to EXP without overflow occuring is 87.3365.

INTRINSIC FUNCTIONS (CONT.)

NAME	EXAMPLE	PURPOSE/USE
FRE(X)	16Ø PRINT FRE(Ø)	Gives the number of memory bytes currently unused by BASIC not including strings.
	165 PRINT FRE(A$)	Gives number of unused memory bytes including strings.
INT(X)	17Ø PRINT INT(X)	Returns the largest integer less than or equal to its argument X. For example: INT(.23)= Ø, INT(7)=7, INT(-.1)=-1, INT(-2)=-2, INT(1.1)=1. The following would round X to D decimal places: INT(X*1Ø↑D+.5)/INT(1Ø↑D +.5)
LOG(X)	18Ø PRINT LOG(X)	Gives the natural (Base E) logarithm of its argument X. To obtain the Base Y logarithm of X use the formula LOG(X)/LOG(Y). 7 = LOG(7)/LOG(1Ø).
PEEK	19Ø PRINT PEEK(I)	The PEEK function returns the contents of memory address I. The value returned will be =>Ø and <=255. If I is 65535 or <-65535 an "ILLEGAL QUANTITY" error will occur. An attempt to read a non-existent memory address will return 255. (see POKE statement)
POS(I)	2ØØ PRINT POS(I)	Gives the current position of the cursor on screen. It is referenced to the left hand margin and has a value of zero if at left margin. See Special Control and Features section.
RND(X)	21Ø PRINT RND(X)	Generates a random number between Ø and 1. The argument X controls the generation 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 value from a timer. X>- generates a new random number between Ø and 1. Note that V-A)*RND(1)+A will generate a random number between A & B.
SGN(X)	22Ø PRINT SGN(X)	Gives 1 if X>Ø, Ø if X=Ø and -1 if X<Ø.
SIN(X)	23Ø PRINT SIN (X)	Gives the sine of the expression X. X is interpreted as being in radians. Note: COS (X)=SIN(X+3.14159/2) and that 1 Radian =18Ø/π degrees=57.2958 degrees; so that the sine of X degrees=SIN(X/57.2958.
SQR(X)	24Ø PRINT SQR(X)	Gives the square root of the argument X. An "ILLEGAL QUANTITY" error will occur if X is less than zero.
TAB(I)	25Ø PRINT TAB(I)	Spaces to the specified position on screen. May be used only in PRINT statements. It specifies the absolute position from the left hand margin where printing is to start. See Special Features and Controls Section.
TAN(X)	26Ø PRINT TAN(X)	Gives the tangent of the expression X. X is interpreted as being in radians.
USR(I)	2ØØX=USR(I)	Calls machine language subroutine at location I.

STRINGS

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$.

String matrices may be dimensioned exactly like numeric matrices. For instance, DIM A$ (1Ø,1Ø) creates a string matrix of 121 elements, eleven rows by eleven columns (row 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.

The total number of characters in use in strings at any time during program execution cannot exceed the amount of string space, or an "OUT OF MEMORY" error will result.

NAME	EXAMPLE	PURPOSE/USE
DIM	25 DIM A$ (1Ø,1Ø)	Allocates space for a pointer, and length for each element of a string matrix. No string space is allocated.
INPUT	4Ø INPUT X$	Reads a string from the user's terminal. 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.
LET	27 LET A$="FOO"+V$	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.
+	3Ø LET Z$=R$+Q$	String concatenation. The resulting string must be less than 256 characters in length of an "STRING TOO LONG" error will occur.
PRINT	6Ø PRINT X$ 7Ø PRINT "FOO"+A$	Prints the string expression on the screen.
READ	5Ø READ X$	Reads a string from DATA statements within the program. Strings do not have to be quotes; but if they are not, they are terminated on a "," or ":" character or end of line and leading spaces are ignored. See DATA for the format of string data.

STRING FUNCTIONS

NAME	EXAMPLE	PURPOSE/USE
ASC(X$)	300 PRINT ASC(X$)	Returns the ASCII numeric value of the first character of the string expression X$. See Appendix K for an ASCII/number conversion table. An "ILLEGAL QUANTITY" error will occur if X$ 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 =>0 and <=255.
FRE(X$)	272 PRINT FRE("")	When called with a string argument, FRE gives the number of free bytes in string space. This equals amount of all free space.
LEFT$(X$,I)	310 PRINT LEFT$(X$,I)	Gives the leftmost I characters of the string expression X$. If I<=0 or >255 an ILLEGAL QUANTITY" error occurs.
LEN(X$)	220 PRINT LEN(X$)	Gives 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<=0 or >255, an "ILLEGAL QUANTITY" error occurs.
MID$(X$,I,J)	340 PRINT MID$(X$,I,J)	MID$ called with three arguments returns a string expression composed of characters of the string expression X$ starting at the I'th character for J characters. If I>LEN(X$), MID$ returns a null string. If I or J<=0 or >255, an "ILLEGAL QUANTITY" error occurs. If J specifies more characters than are left in the string, all characters from the I'th on are returned.
RIGHT$(X$,I)	320 PRINT RIGHT$(X$,I)	Gives the rightmost I characters of the string expression X$. When I<=0 or >255 an "ILLEGAL QUANTITY" error will occur. If I>=LEN(X$) 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".
VAL(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

"Control" characters are indicated by a super-scripted "C" such as G^{C}. They are obtained by holding down the CTRL key while typing the specified letter. Control characters are NOT displayed on the TV screen. B^{C} and C^{C} must be followed by a carriage return. Screed editing characters are indicated by a sub-scripted "E" such as D$_{E}$. They are obtained by pressing and releasing the ESC key then typing specified letter. Edit characters send information only to display screen and does not send data to memory. For example, U^{C} moves to cursor to right and copies text while A$_{E}$ moves cursor to right but does not copy text.

CHARACTER	DESCRIPTION OF ACTION
"RETURN" key	The "RETURN" key must end every line that is typed in to tell the APPLE II that you have finished the line.
: (Colon)	A colon may be used to separate statements or a line. Colons may be used in direct or indirect statements. The only limit to the number of statements per line is that the total number of characters including spaces may not exceed 255.
? (Question Mark)	Question marks are equivalent to "PRINT" command. 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 displayed as 10 PRINTX.
"RESET Key	Immediately interrupts any program execution and resets computer. Also sets all text mode with scrolling window at maximum. Control is transfered to System Monitor and APPLE prompts with a "*" (asterisk) and a bell. Hitting RESET key does NOT destroy existing BASIC or machine language program. From the System Monitor, user machine language programs may be typed in. From the Monitor, you may return to APPLESOFT BASIC without destroying current user BASIC program by typing "ØG" and depressing the RETURN key. If while you are in the Monitor, you change any data in the range $Ø.1FF, you can return to APPLESOFT by typing in "274DG" and this will kill any current user BASIC program and will re-write locations Ø.1FF.
B^{C}	If in System Monitor (as indicated by a "*", prompt character and a flashing cursor), a control-B and a carriage return will transfer control to BASIC, scratching (killing) APPLESOFT and any existing BASIC program. It will set HIMEM: to maximum installed user memory and LOMEM: to 2048.
C^{C}	If in APPLESOFT BASIC, halts program and displays line number where stop occured. Program may be continued with a CONT command. If in System Monitor, (as indicated by "*"), control C and a carriage return will enter integer BASIC killing APPLESOFT BASIC and the user program.
G^{C}	Sounds bell (beeps speaker)
H^{C}	Backspaces cursor and deletes any overwritten characters from computer but not from screen. APPLE supplied keyboards have a special "←" on the right side of the keyboard that provides this function without using the control button.
J^{C}	Issues line feed only
V^{C}	Compliment to H^{C}. Forward spaces cursor and copies overwritten characters. APPLE keyboards have "→" key on right side which also performs this function.
X^{C}	Immediately deletes current line.
A$_{E}$	Move cursor to right; does not copy any data
B$_{E}$	Move cursor to left; does not copy any data
C$_{E}$	Move cursor down; does not copy any data
D$_{E}$	Move cursor up; does not copy any data
E$_{E}$	Clear text from cursor to end of line
F$_{E}$	Clear text from cursor to end of page
@$_{E}$	Home cursor to top of page, clear text to end of page.

Special Controls and Features

BASIC Example	DESCRIPTION
10 POKE-16304,Ø	Switches display mode from text mode to color graphics without clearing screen to black. ("PLTG" command switches to color and clears screen to black and sets mixed mode.)
20 POKE-16303,Ø	Switches display from color graphics to all text mode without resetting scrolling window. ("TEX" command also resets scrolling window to maximum and positions cursor in lower left hand corner of TV display.
30 POKE-16302,Ø	Sets all color graphics mode of 40x48 grid; i.e., no text at bottom of screen
40 POKE-16301,Ø	Sets mixed color graphics mode; i.e., 40x40 grid of 16 colors with four lines of text each 40 characters at bottom of screen. (Automatically done by a "PLTC" command.)
50 POKE 32, L	Set left margin of TV display to value specified by L in the range of Ø to 39 where Ø is left most position.
60 POKE 33, W	Set the width (number of characters per line) of TV display to the value specified by W. W must be greater than zero. Wth must be less than 40; i.e., the right margin must be 39 or less.
70 POKE 34, T	Set top margin line of TV display to value specified by T in the range of Ø to 23 where Ø is the first line on the screen. A POKE 34,4 will not allow text to be outputted to the first four lines on the screen.
80 POKE 35, B	Set bottom margin line of TV display to value specified by T in the range of Ø to 23. B must also be larger than T above; i.e., the bottom of the display cannot be above the top. Text will scroll up when last line is reached.
90 CH=PEEK(36)	Read back the current horizontal position of the cursor and set variable CH equal to it. CH will be in the range of Ø to 39 and is a relative position referenced to the left hand margin as set by POKE 32,L. Thus, if the margin was set by POKE 32,5, then the left margin is 6 characters from the left edge of the screen and if PEEK (36) returned a value of 5 then the cursor was 11 character positions from the left edge of the screen and 6 characters from the left margin. This is identical to the "POS(X)" function where X is a dummy variable (See next example.)
100 POKE 36,CH	Move the cursor to a position that is CH+1 character positions from the left hand margin. (Exp: POKE 36,0 ill cause next character outputted to be at left margin). If left margin was set at 6 (POKE 32,6) and you wanted to provide a character three positions from left edge, then the left margin must be changed prior to outputting. CH must be less than or equal to the window width as set by POKE 22,W and must be greater than or equal to zero.
110 CV=PEEK(37)	Read back the current vertical position of the cursor and set CV equal to it. CV is the absolute vertical position of the cursor and is not referenced to the top or bottom of page settings. Thus CV=Ø is top line on screen and CV=23 is bottom. The value of CV will be between T (top) and B (bottom).
120 POKE 37,CV	Move the cursor to the absolute position specified by CV and CV is greater than or equal to T and less than or equal to B. Ø is the top most line and 23 is the last line.
130 POKE 50,255	Set text mode output to be white characters on black background (Normal mode).
140 POKE 50,127	Set text mode to flash outputted characters.
150 POKE 50,63	Set text mode output to be black characters on white background. (Inverse mode.)

Special Controls and Features (Cont.)

BASIC Examples	DESCRIPTION
160 X=USR(-936)	Home cursor; i.e., move cursor to top-left most edge of screen as defined by window settings (L,W,T,B), then clear all characters inside window. Characters outside defined window are not affected. This is same as $@_E$ (Escape E).
170 X=USR(-958)	Clear inside of window from current cursor position to bottom margin and left margin. Characters to the left or above the cursor will not be affected. This is the same as F_E (Escape F).
180 X=USR(-868)	Clear current line from cursor to right margin. This is the same as E_E (Escape E).
190 X=USR(-922)	Issues a line feed to the TV display.
200 X=USR(-912)	Scrolls up text one line; i.e., moves each line of text within the defined window up one position. Old top line is lost; old second line becomes line one; bottom line is now blank. Characters outside defined window are not affected.
210 X=USR(-1233)	Resets window to maximum and sets all text mode. X=USR(-1233) is the same as POKE 32,0: POKE 33,39: POKE 34,0: POKE 35,23: POKE-16303,0: POKE 36,0: POKE 37,23.
220 X=PEEK(-16336)	Toggle speaker once.
230 X=PEEK(-16384)	Read keyboard; if X>127 than key was depressed and X AND 127 is ASC II value of key depressed. This is useful in long programs to have the computer check to see if the user wants to interrupt with new data without stopping program execution.
240 X=POKE(-16368,0)	Reset keyboard strobe so that next character may be read in.
250 X=PEEK(-16287)	Read paddle #0 push button switch. If X>127 then paddle button is depressed.
260 X=PEEK(-16286)	Same as above but paddle #1
270 X=PEEK(-16285)	Paddle #2 pushbutton.
280 X=POKE-16296,1	Set Game I/O output #Ø to TTL high (3.5 volts).
290 X=POKE-16295,0	Set Game I/O output #0 to TTL low (0.3 volts).
300 X POKE-16294,1	Set Game I/O output #1 to TTL high (3.5 volts).
310 X=POKE-16293,Ø	Set Game I/O output #1 to FTL low (0.3 volts).

A P P E N D I C E S

APPENDIX A

Getting APPLESOFT Floating Point BASIC Up

Unlike APPLE integer BASIC, which is always "in" the computer's permanent ROM memory, APPLESOFT BASIC must be loaded from cassette tape into the computer each time you wish to use it (because it resides in RAM, it is lost when power is turned off). Since APPLESOFT BASIC occupies approximately 10k bytes of memory, a computer with 16k bytes or more memory is required to use APPLESOFT BASIC.

APPLESOFT BASIC is entered into the computer just like any BASIC program - simply type: LOAD
start the tape
depress the RETURN key

After about 1½ minutes APPLESOFT will have loaded, and a ">" prompt character followed by a cursor will be displayed.

Typing "RUN" as you always do to run a program will produce the following message:

```
                APPLE COMPUTER
EXTENDED PRECISION FLOATING POINT BASIC
INSTRUCTIONS WILL BE LOST AFTER ENTERING
FP BASIC
INSTRUCTIONS (Y/N) ?
```

Don't answer the question quite yet...APPLESOFT BASIC requires 10k bytes of memory, which leaves 5k bytes free for your programs (in a 16k system). When you load the APPLESOFT tape, however, there is an additional 5k byte program which loads along with APPLESOFT BASIC. This program will (when APPLESOFT BASIC is run for the first time after loading) provide instruction, a summary of commands, and allow you to choose between including GRAPHIC commands or the commands LET and REM in the BASIC. This preliminary program will be automatically erased after it has been used, leaving the memory it previously occupied free for user programs.

So, in answer to the question "INSTRUCTIONS (Y/N) ?" type Y or YES and hit RETURN.

You will then be asked "SUMMARY OF COMMANDS (Y/N) ?" type Y or YES as above.

The screen will now be filled with a partial list of the commands used in APPLESOFT BASIC. Just hit the space bar to see more on the next page. If you want to return to the beginning of this program (the question INSTRUCTIONS (Y/N) ?) then just depress the "ESC" key.

This is sometimes useful when you want to refer back to a previous page of information.

After 12 pages of information, you will reach a page with information about the prompt character. When you are familiar with the commands used in APPLESOFT BASIC, you can answer NO to the question "SUMMARY OF COMMANDS (Y/N) ?" and jump immediately to this page, bypassing all the command information.

AN IMPORTANT NOTE: One of the functions of the prompt character, besides PROMPTing you for input to the computer, is to identify at a glance which language the computer is programmed to respond to at that time. For instance, up till now you have seen two prompt characters:

"*" for the MONITOR (when you hit RESET)

">" for APPLE BASIC (the normal integer BASIC)

and now we introduce a third:

" " for APPLESOFT floating point BASIC.

By simply looking at this prompt character, you can easily tell (if you forget) which language the computer is in.

ANOTHER IMPORTANT NOTE: If you accidently hit RESET and are in the MONITOR (as shown by the "*" prompt character), you may be able to return to APPLESOFT BASIC, with the BASIC and your program intact by typing "ØG" and depressing the "RETURN" key. If this does not work, you will have to re-load APPLESOFT from cassette tape. Also, typing Control-C or Control-B from the monitor will transfer you to APPLE integer BASIC and erase APPLESOFT BASIC.

Next, you will reach the last page of questions and information in this preliminary program. We could have gotten here immediately from the first question INSTRUCTIONS (Y/N)? just by typing NO (actually, just hitting RETURN also does the trick).

It reads: Option #1 - Graphic commands but no LET or REM commands.

or

Option #2 - LET & REM commands but no graphic commands.

Select Option 1 if you are going to try the examples in the first part of this manual. Do it by answering the question "Which Option" by typing a "1" and depressing the "RETURN" key. After a few seconds the preliminary program will be erased. APPLE will then display copy-rights information and ask for "Memory Size?" which most will just depress the "RETURN" key and let APPLE set memory to maximum. APPLE will now respond with "XXXXX Bytes Free" to inform you how much space is available for your program and then will display a "]" prompt character and a flashing cursor. APPLESOFT floating point BASIC is now "on the air".

APPENDIX B

PROGRAM EDITING WITH APPLESOFT BASIC

Most ordinary humans make mistakes occassionally.... especially when writing computer programs. To facilitate correcting these "oversights" Apple has incorporated a unique set of editing features into APPLESOFT BASIC.

To make use of them you will first need to familiarize yourself with the functions of four special keys on the Apple II keyboard. They are: (Escape), → (Right Arrow) ←(Left Arrow), and REPT (Repeat).

ESC

The escape key ("ESC") is the leftmost key in the second row from the top. It is ALWAYS used with another key (such as A, B, C or D keys) ie. using the escape key requires you to push and release "ESC" then push and release A etc....alternately.

This operation or sequence of the "ESC" key and another key is written as subscript E (A_E) and is read "Escape-A". There are four escape functions used for editing:

A_E - "escape-A" moves cursor to the right
B_E - "escape-B" moves cursor to the left
C_E - "escape-C" moves cursor down
D_E - "escape-D" moves cursor up

Using the escape key and the desired key, the cursor may be moved to any location on the screen without affecting anything that is already displayed there.

RIGHT HAND ARROW (→)

The right arrow key (→) moves the cursor to the right. It is the most time saving key on the keyboard because it not only moves the cursor, but,

IT COPIES ALL CHARACTERS AND SYMBOLS which IT "MOVES ACROSS", INTO APPLE II'S MEMORY, JUST AS IF YOU HAD TYPED THEM IN FROM THE KEYBOARD YOURSELF!

LEFT HAND ARROW (←)

The left arrow key (←) moves the cursor to the left. It removes all characters and symbols it "moves across" from Apple II's memory but not from the TV display. It is similar in use to the backspace key on standard typewriters.

REPT

The "REPT" key is used with another character key on the keyboard. It causes a character to be repeated as long as the REPT key is held down.

Now you're ready to use these edit functions to save time when making changes or corrections to your program. Here are a few examples of how to use them.

Example 1 - Fixing typos

Suppose you've entered a program by typing it in, and when you run it, the computer prints SYNTAX ERR and stops, presenting you with the "]" prompt and the flashing cursor.

Enter the following program and "RUN" it. Note that "PRIMT" and "PREGRAM" are mis-spelled on purpose. Below is how it will look on your TV display.

```
]10 PRIMT "THIS IS A PREGRAM"
]20 GOTO 10
]RUN

 ?SYNTAX ERR IN  10
```

Now type in "LIST" as below:

```
]LIST

  10 PRIMT "THIS IS A PREGRAM"
  20 GOTO 10
]
```

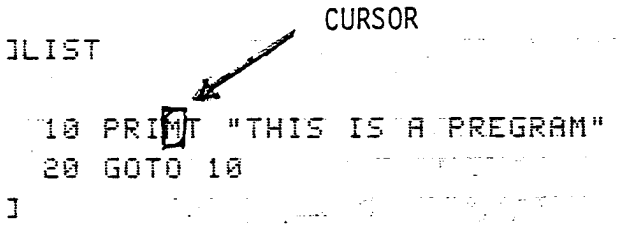
CURSOR

To move the cursor up to the error in line 10, type escape-D twice. The TV display will now look like this:

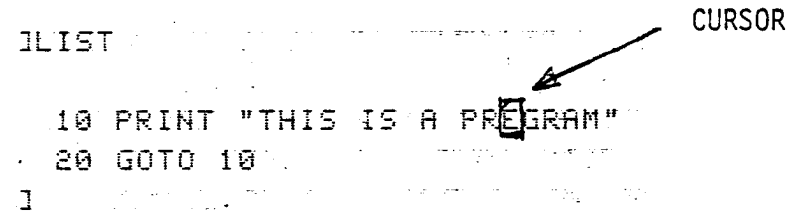
```
]LIST
                    CURSOR
  10 PRIMT "THIS IS A PREGRAM"
  20 GOTO 10
]
```

Now hit the right arrow (→) 6 times to move the cursor on to the "M" in "PRIMT". Remember, using the right arrow copies

all characters covered into Apple's memory just as if you were typing them in from the keyboard. The TV display will now look like this:



Now type the letter "N" to correct the spelling of "PRIMT", then copy (using the "→" key and the "REPT" key) over to the letter "E" in "PREGRAM". The TV screen will now look like this:



If you typed too many "→"'s by holding down the "REPT" key too long, use the "←" key to backspace back to the "E". Now, type the letter "O" to correct "PREGRAM" and copy using the "→" key to the end of line 10. Note that unnecessary spaces at the end of a program line will slow your program down so make sure that you hit the "RETURN" key just after the last character in a line.

Type "LIST" to see your corrected program:

```
]LIST

  10 PRINT "THIS IS A PROGRAM"
  20 GOTO 10
]
```

Now "RUN" it (Use a control-C to stop the program):

```
]RUN
THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM
 THIS IS A PROGRAM

BREAK IN 10
]
```

Example 2 - Inserting text into an existing line

Suppose in the previous example, you wanted to insert a "TAB(X)" command after the "PRINT" in line 10. Here's how. First "LIST" the line to be changed:

```
]LIST 10

  10 PRINT "THIS IS A PROGRAM"
]
```

CURSOR

Type escape - D until the cursor is on the line to be changed (in this case only ~~one~~ DE ~~is~~ required); then use the "→" and "REPT" keys to copy over to the first quotation mark. Your TV display should now look like this:

CURSOR

```
]LIST 10

  10 PRINT "THIS IS A PROGRAM"
]
```

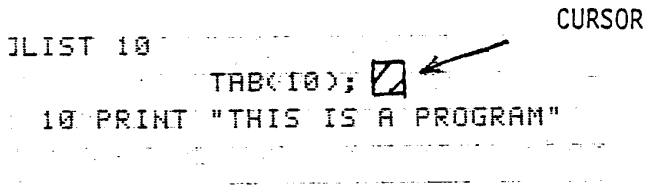
Now type another escape - D to move the cursor to the line just above the current line and the display will look like:

CURSOR

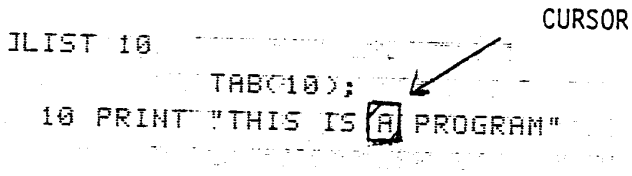
```
]LIST 10

  10 PRINT "THIS IS A PROGRAM"
```

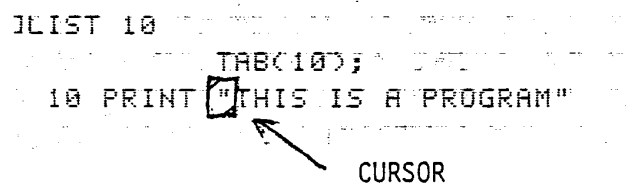
Type in the message to be inserted which, in this case, is "TAB(1Ø);". Your TV display should now look like this:



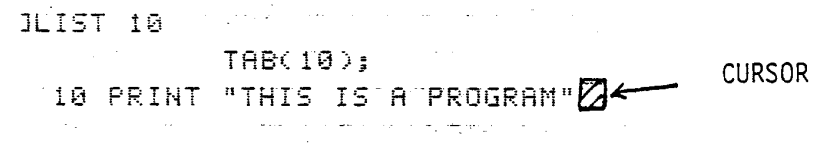
Type an escape - C to move the cursor down one line so that the display looks like this:



Now backspace back to the first quotation mark using escape - B (~~or the "←" key~~). The TV display will now look like this:



From here, copy the rest of the line using the "→" and "REPT" keys until the display looks like this:



Depress the "RETURN" key and type "LIST" to get the following:

```
]LIST

  10 PRINT TAB(10);"THIS IS A PROGRAM"
  20 GOTO 10
]
```

Remember, using the escape keys, one may copy and edit text that is displayed anywhere on the TV display.

APPENDIX C

ERROR MESSAGES

After an error occurs, BASIC returns to command level as indicated by "]" prompt character and a flashing cursor. Variable values and the program text remain intact, but the program can not be continued and all GOSUB and FOR loop counters are set to Ø.

When an error occurs in a direct statement, no line number is printed.

Format of error messages:

Direct Statement	?XX ERR
Indirect Statement	?XX ERR IN YY

In both of the above examples, "XX" will be the error code. The "YY" will be the line number where the error occured for the indirect statement. Error messages for indirect statements will be not outputted until a "RUN" is executed.

The following are the possible error codes and their meanings.

ERROR MESSAGE	MEANING
CAN'T CONTINUE	Attempt to continue a program when none exists, an error occured, or after a new line was typed into the program.
DIVISION BY ZERO	Dividing by zero is an error.
ILLEGAL DIRECT	You cannot use an INPUT or DEFEN statement as a direct command.
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 (LET A (-1=0) b) an unreasonably large matrix subscript (>65535) c) LOG-negative or zero argument d) SOR-negative argument

ERROR MESSAGES (cont.)

ERROR MESSAGE	MEANING
ILLEGAL QUANTITY (cont)	e) A↑B with A negative and B not an integer. f) use of MID$, LEFT$, RIGHT$, WAIT, PEEK, POLE, TAB, SPC or ON.. GOTO with an improper argument.
NEXT WITHOUT FOR	The variable in a NEXT statement corresponds to no previously executed FOR statement.
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.
OUT OF MEMORY	Program too large, too many variables, too many FOR loops, too many GOSUB's, too complicated an expression or any combination of the above.
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.
REDIM'D 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.
RETURN WITHOUT GOSUB	A RETURN statement was encountered without a previous GOSUB statement being executed.

ERROR MESSAGES (cont.)

ERROR MESSAGE	MEANING
STRING TOO LONG	Attempt was made by use of the concatenation operator to create a string more than 255 characters long.
BAD SUBSCRIPT	An attempt was made to reference a matrix element which is outside the dimensions of the matrix. 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).
SYNTAX ERROR	Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.
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.
UNDEF'D STATEMENT	An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist.
UNDEF'D FUNCTION	Reference was made to a user defined function which had never been defined.

The line which the error occurs on will be listed after the error message.

APPENDIX D

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 0, maximum is 65529), it takes the same number of bytes. Putting as many statements as possible on a line will cut down on the number of bytes used by your program.

2) Use integer as opposed to real matrixes where ever possible.

3) 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 nonblank character are ignored.

4) Delete all REM statements. Each REM statement uses at least one byte plus the number of bytes in the common 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.

5) 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 improvement.

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

7) 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. Of, if you are asking the

terminal 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.

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

9) Use the zero elements of matrices; for instance, A(0), B(0,X).

STORAGE ALLOCATION INFORMATION

Simple real or integer (non-matrix) numeric variables like V use 7 bytes; 2 for the variable name, and 5 for the value. Simple non-matrix string variables also use 6 bytes; 2 for the variable name, 2 for the length, and 2 for a pointer.

Real matrix variables use a minimum of 13 bytes. Two bytes are used for the variable name, two for the size of the matrix, two for the number of dimensions and two for each dimension along with five bytes for each of the matrix elements. Integer (AB% (X,Y...)) matrix variables use only 2 bytes for each matrix element.

String variables also use one byte of string space for each character in the string. This is true whether the string variable is a simple string variable like A$, or an element of a string matrix such as Q1$(5,2).

When a new function is defined by a DEF statement, 6 bytes are used to store the definition.

Reserved words such as FOR, GOTO or NOT, and the names or the intrinsic functions such as COS, INT and STR$ take up only one byte of program storage. All other characters in programs use one byte of program storage each.

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

1) Each active FOR...NEXT loop uses 16 bytes.

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

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

APPENDIX E

Speeding Up Your Program

The hints below 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) 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 that it does to fetch the value of a simple or matrix variable. This is especially important within FOR...NEXT loops or other code that is executed repeatedly.

2) 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.

3) NEXT statements without the index variable. NEXT is somewhat faster than NEXT I 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.

4) Delete all unnecessary 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.

5) During program execution, when APPLESOFT encounters a new line reference such as "GO TO 1000" it scans the entire user program starting at the lowest line until it finds the referenced line number (1000 in this example). Therefore frequently referenced lines should be placed as early in the program as possible. For example:

0 GO TO 1000	Skip over subroutines, etc.
10.....999	Frequently referenced statements such as DATA, GOSUB's, etc.
1000....1099	Define most fequently used variables and constants as variables.
1100....1199	User defined subroutines "DEF FN".
1200....1299	Dimension matrices
2000....	Main program begins.

APPENDIX F

DERIVED FUNCTIONS

The following functions, while not intrinsic to APPLESOFT BASIC, can be calculated using the existing BASIC functions and can be easily implimented by using "DEF FN" function.

FUNCTION	FUNCTION EXPRESSED IN TERMS OF BASIC FUNCTIONS
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
CONTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SOR(-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

APPENDIX G

CONVERTING BASIC PROGRAMS NOT WRITTEN FOR APPLESOFT

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 Apple II.

1) Matrix subscripts. Some BASIC's use " [" and "] " to denote matrix subscripts. APPLESOFT BASIC uses " (" and ") ".

2) Strings. A number of BASIC's 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 BASIC's, a declaration of the form DIM A$(I,J) declares a string matrix of J elements each of which has a length I. Convert DIM statements of this type to equivalent ones in APPLESOFT BASIC: DIM A$(J).

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

APPLESOFT BASIC uses LEFT$, RIGHT$ and MID$ to take substrings of strings. Other BASIC's 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 BASIC's allow statements of the form: 500 LET B=C=0. This statement would set the variables B & C to zero.

In APPLESOFT 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 BASIC's 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 BASIC's will have to be re-written using FOR...NEXT loops to perform the appropriate operations.

APPENDIX H

ASCII CHARACTER CODES

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
Ø	NUL	32	SPACE	64	@
1	SOH	33	!	65	A
2	STX	34	"	66	B
3	ETX	35	#	67	C
4	EOT	36	$	68	D
5	ENQ	37	%	69	E
6	ACK	38	&	7Ø	F
7	BEL	39		71	G
8	BS	4Ø	(	72	H
9	HT	41	)	73	I
1Ø	LF	42	*	74	J
11	VT	43	+	75	K
12	FF	44	,	76	L
13	CR	45	-	77	M
14	SO	46	.	78	N
15	SI	47	/	79	O
16	DLE	48	Ø	8Ø	P
17	DC1	49	1	81	Q
18	DC2	5Ø	2	82	R
19	DC3	51	3	83	S
2Ø	DC4	52	4	84	T
21	NAK	53	5	85	U
22	SYN	54	6	86	V
23	ETB	55	7	87	W
24	CAN	56	8	88	X
25	EM	57	9	89	Y
26	SUB	58	:	9Ø	Z
27	ESCAPE	59	;	91	
28	FS	6Ø		92	
29	GS	61	=	93	
3Ø	RS	62		94	
31	US	63	?	95	

LF=LINE FEED CR=CARRIAGE RETURN

CHR$ is a string function which returns a one character string which contains the ASCII equivalent of the argument, according to the conversion table above. ASC takes the first character of a string and converts it to its ASCII decimal.

One of the most common uses of CHR$ is to send a special character to the user's terminal. The most often used of these characters is the BELL (ASCII 7). Printing this character will cause a "beep". This may be used as a preface to an error message, as a novelty, or just to wake up the user if he has fallen asleep. (Example: PRINT CHR$(7);)

APPENDIX I

MEMORY MAP - APPLE II with APPLESOFT BASIC LOADED

MEMORY RANGE*	DESCRIPTION
Ø.1FF	Program work space; not available to user.
2ØØ.2FF	Keyboard character buffer.
3Ø8.3FF	Available to user for short machine language programs.
4ØØ.7FF	Screen display area for text or color graphics.
8ØØ.29FF	APPLESOFT BASIC complier.
2AØØ.XXX	User program and variables where XXX is maximum available RAM memory to be used by APPLESOFT. This is either total system RAM memory or less if the user is reserving part of high memory for machine language routines.
CØØØ.CFFF	Hardware I/O addresses.
DØØØ.ØFFF	Future ROM expansion
EØØØ.F7FF	Apple Integer BASIC
F8ØØ.FFFF	Apple System Monitor

* Numbers are in hexidecimal notation.

Appendix J

Literature References

Ahl, David (Editor), The Best of Creative Computing Vol I.
Morristown, NJ: Creative Computing Press, 1977.

_____, The Best Of Creative Computing Vol II.
Morristown, NJ: Creative Computing Press, 1977.

Albrecht, Robert, My Computer Likes Me when I speak in BASIC.
Menlo Park, CA: Dymax, 1972.

_____, Leroy Finkel, and Jerry Brown, BASIC.
New York: John Wiley & Sons, Inc., 1973.

Aribib, Michael A., Brains, Machines, and Mathematics.
New York: McGraw-Hill, 1977.

Bergman, Samuel and Steven Bruckner, Introduction to Computers and Computer Programming.
Reading, Mass: Addison-Wesley Publishing Co., 1972.

Brand, Stewart, II Cybernetic Frontiers.
New York, Random House, 1975.

Brown, Jerald, R., Instant Basic.
Menlo Park, CA: Dymax, 1977.

Brown, R. W., Basic Software Library.
Crofton, Md: Scientific Research Inst., 1976.

Clarke, Sheila, "The Remarkable Apple Computer"
Kilobaud, 1977, 2:34,38.

Coan, James, S., Basic Basic.
Rochelle Park, NJ: Hayden Book Company, Inc., 1970.

_____, Advanced Basic.
Rochelle Park, NJ: Hayden Book Company, Inc., 1977.

Crowley, Thomas, H., Understanding Computers.
New York: McGraw-Hill, 1977.

Feldman, Phil, and Tom Rugg, "Hangmath!" Kilobaud.
1977, 4:112-115.

Fenichel, Robert, R., and Joseph Weizenbaum (Introductions),
Readings from Scientific American: Computers and Computation.
San Francisco: W. H. Freeman and Company, 1971.

Gruenberger, Fred, and George Jaffray, Problems For Computer Solution.
New York: John Wiley & Sons, Inc., 1965.

Hellerman, H., Digital Computer Systems Principles, 2nd Ed.
New York: McGraw-Hill, 1973.

Jordan, P., Condensed Computer Encyclopedia.
New York: McGraw-Hill, 1969.

Kemeny, John, G. and Thomas E. Kurtz, Basic Programming.
New York: John Wiley & Sons, Inc., 1971.

Koberg, Don, and Jim Bagnall, The Universal Traveler.
Los Altos, CA: Eillism Kaufmann, Inc., 1976.

Kohl, Herbert, R., Math, Writing, & Games.
New York: Vintage Books, 1974.

La Fave, L., G. Milbrandt, and D. Garth, Problem Solving:
The Computer Approach.
New York: McGraw-Hill, 1973.

Ledgard, Henry, F., Programming Proverbs.
Rochelle Park, NJ: Hayden Book Company, Inc., 1975.

Lehman, John A., "A Small Business Accounting System."
Byte, 1976, 10:8-12

McCabe, Dwight, PCC's Reference Book.
Menlo Park, CA: People's Computer Company, 1977.

Nilsson, N., Artificial Intelligence.
New York: McGraw-Hill, 1971.

Poole, Lon, and Mary Borchers, Some Common Basic Programs.
Berkeley: Adam Osborne & Associates, Inc., 1977.

Rugg, Tom, and Phil Feldman, "A Useful Loan Payment Program."
Kilobaud, 1977, 2:68-69.

_____, "BASIC Timing Comparisons."
Kilobaud, 1977, 6:66-70.

Sharpe, William, F., and Nancy L. Jacob, BASIC.
New York: The Free Press, 1971.

Smith, Robert, E., Discovering Basic.
Rochelle Park, NJ: Hayden Book Company, Inc., 1970.

Tausworthe, Robert C., Standarized Development of Computer Software.
Englewood Cliffs, NJ: Prentice-Hall, 1977

Technica Education Corporation, Teach Yourself Basic, Vol. I.
Salt Lake City, 1970.

_____, Teach Yourself Basic, Vol. II.
Salt Lake City, 1970.

Warren, Jim, C., (editor), The First West Coast Computer Faire
Conference Proceedings.
Palo Alto, CA: Computer Faire, 1977.

Warren, Carl Denver, "Simplified Billing System"
Kilobaud, 1977, 6:94-95.

White, James, Your Home Computer.
Menlo Park, CA: Dymax, 1977.

Wilkinson, Lee, "Cure Those End-of-the Month Blues."
Kilobaud, 1977, 2:34-35.

Wozniak, Stephen, "The Apple-II."
Byte, 1977, 2(5): 34-44.