Apple Software Bank

APPLESOFT™

Reference Manual

APPLESOFT™ II
REFERENCE MANUAL

Extended Precision Floating Point
BASIC Language

August 1978

Reorder Apple Part No. A2L0006X

APPLESOFT™ II
Table of Contents

Stock #030-0013-02

INTRODUCTION

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

This BASIC is intended for use in business, scientific 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 J.

If your Apple II does not have Floating Point BASIC loaded and running, follow the procedures in Appendix 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          3Ø
```

As you can see, BASIC can do division and multiplication as well as subtraction. Note how a "," (comma) was used in the print command to print two values instead of just one. The comma divides the 40 character line into 3 columns, each 16 characters wide. The result is that a "," causes BASIC to skip to the next 16 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 63999.

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, APPLESOFT 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 each statement 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 automatically 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 $1.0*10^{38}$, while the smallest positive number is $1.0*10^{-39}$.

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

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

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

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

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

Scientific notation is formatted as follows: SX.XXXXXXXXESTT (each X being an integer 0 to 9)

The leading "S" is the sign of the number, nothing 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 10 raised to the power of the number to the right of the "E".

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 1Ø 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

APPLE will reply with:

]

Color Graphics Example

Now type in:

GR

This will black out the top twenty lines of text 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:

COLOR = 13

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

Now type in:

PLOT 2Ø, 2Ø

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

Now type in:

HLIN Ø,3Ø AT 2Ø

*If the square is not yellow, your color TV is not tuned properly: adjust the tint & color controls to achieve a clear lemon yellow.

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:

```
COLOR = 6
```

To change to a new color and then type in:

```
VLIN 1Ø,39 AT 3Ø
```

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

```
TEXT
```

The character display on the screen is Apple's way of showing 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           .333333333
```

Print Format

As explained earlier, including a " , " in a print statement causes it to space over to the next sixteen 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. Try it.

Try the following examples:

```
A)  PRINT 1,2,3
     1              2               3

B)  PRINT 1;2;3

     123

C)  PRINT -1;2;-3

     -12-3
```

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
$1 \times 1Ø^{2Ø}$	1E+2Ø
$-12.34567896 \times 1Ø^{1Ø}$	-1.2345679E+11
1ØØØØØØØØØ	1E+9
999999999	999999999

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 (40 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
 314.159
```

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 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
 314.159
```

```
? 3
 28.27431
? 4.7
 69.3977231
?
```

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 control C and 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 unless they contain a reserve word from the list below. 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$	RGOTO (variable names cannot contain reserved words)
COUNT	
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 CALL CHR$ CLEAR COLOR= CONT COS DATA DEF

DEL DIM DRAW END EXP FLASH FN FOR FRE GET GOSUB GOTO GR

HCOLOR= HIMEM: HGR HGR2 HLIN HOME HPLOT HTAB IF IN# INPUT INT

INVERSE INII LEFT$ LEN LET LIST LOAD LOMEM: LOG MID$ NEW

NEXT NORMAL NOT NOTRACE ON ONERR OR OUT PDL PEEK PLOT

POKE POP POS PRINT PR# READ RECALL REM RESTORE RESUME RETURN

RIGHT$ RND ROT=RETURN RUN SAVE SCALE=SCRN(SGN SHLOAD SIN SPC(

SPEED SQR STEP STORE STOP STR$ TAB(TAN TEXT THEN TO TRACE

VAL VLIN VTAB USR WAIT

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

PRINT Z, Z-A
 7             2
```

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

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

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 Control-C and 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  GR
2Ø  NX = Ø:NY = Ø:X = Ø:Y = 5:XV = 2:YV = 1
3Ø  T9 = 39:TØ = Ø:J = 1:K = 25Ø
4Ø  NX = X + XV:NY = Y +YV
5Ø  IF NX>  = T9 THEN NX =T9
6Ø  IF NX<  = TØ THEN NX =TØ
7Ø  IF NY>  = T9 THEN NY =T9
8Ø  IF NY<  = TØ THEN NY =TØ
9Ø  IF NX = T9 OR NX = TØ THEN XV = -XV
1ØØ IF NY = T9 OR NY = TØ THEN YV = -YV
11Ø COLOR = 13: PLOT NX, NY
12Ø COLOR = Ø: PLOT X,Y
13Ø X = NX:Y = NY
14Ø I = I + J: IF I < K THEN 4Ø
15Ø TEXT : 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 <u>greater than</u> or equal to the final value. The reason is that we are now counting by a negative number. In the previous examples it was the opposite, so we were checking for a variable <u>less than</u> or equal to the final value.

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

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

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

```
10  FOR I=1 TO 5
20  FOR J=1 TO 3
30  PRINT I,J
40  NEXT 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 0 to 15. Matrix subscripts always start at 0; therefore, in the above example, we have allowed for 16 numbers in matrix A.

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

As an example of how matrices are used, try the following 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=0
 80  FOR I=1 TO 7
 90  IF A(I)<=A(I+1) THEN 140
100  T=A(I)
110  A(I)=A(I+1)
120  A(I+1)=T
```

```
130  F=1
140  NEXT I
150  IF F=1 THEN 70
160  FOR I=1 TO 8
170  PRINT A(I)
180  NEXT I
```

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

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

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 FIRST 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 FIRST 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 4Ø 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 1Ø for another guess, we need to make the READ begin with the first piece of data again. This is the function of the "RESTORE". After the RESTORE is encountered, the next piece of data read will be the first piece in the first DATA statement again.

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

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 1Ø 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 <u>exact</u> number that you expected.

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*1Ø^D+.5)/INT(1Ø^D+.5)
```

In this case, 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=1Ø is one place; D=1ØØ, 2 places; D=1ØØØ, 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.

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 numeric variables, string variables can be assigned specific values. String variables are distinguished from numeric variables by a "$" after the variable name.

For example, try the following:

```
A$= "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 a "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
03
```

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         3.141
```

"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)
 5
```

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.

```
1ØØ DIM A$(15)
11Ø FOR I=1 TO 15:READ A$(I):NEXT I:
12Ø F=Ø:I=1
13Ø IF A$(I)<=A$(I+1) THEN 18Ø
14Ø T$=A$(I+1)
15Ø A$(I+1)=A$(I)
16Ø A$(I)=T$
17Ø F=1
18Ø I+1: IF I <= 15 GOTO 13Ø
19Ø IF F THEN 12Ø
2ØØ FOR I=1 TO 15:PRINT A$(I);NEXT I
22Ø DATA APPLE,DOG,CAT,RANDOM,COMPUTER,BASIC
23Ø DATA MONDAY,"***ANSWER***","FOO: "
24Ø 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.

```
10  GR : REM  INITIALIZE COLOR GR
     APHICS; SET 40X40 TO BLACK.
     SET WINDOW TO 4 LINES AT BOT
     TOM
20  HOME : REM  CLEAR ALL TEXT AT
     BOTTOM
30  COLOR= 2: PLOT 0,0: REM   MA
     GENTA SQUARE AT 0,0
40  LIST 30: GOSUB 1000
50  COLOR= 1: PLOT 39,0: REM  BLU
     E SQUARE AT X=39,Y=0
60  HOME : LIST 50: GOSUB 1000
70  COLOR= 12: PLOT 0,39: REM  GR
     EEN SQUARE ATX=0, Y=39
80  HOME : LIST 70: GOSUB 1000
90  COLOR= 9: PLOT 39,39: REM  OR
     ANGE SQUARE AT X=39, Y=39
100  HOME : LIST 90: GOSUB 1000
110  COLOR= 12: PLOT 19,19: REM
     YELLOW SQUARE AT CENTER SCRE
     EN
120  HOME : LIST 110: GOSUB 1000
130  HOME : PRINT "PLOT YOUR OWN
     POINTS"
140  PRINT "REMEMBER X&Y MUST BE
     >=0 & <=39"
150  INPUT "ENTER X,Y ";X,Y
160  COLOR= 8: PLOT X,Y
170  PRINT "TYPE 'CTRL C' AND HIT
      RETURN TO STOP": GOTO 150
180  STOP
1000  PRINT "***HIT ANY KEY TO CO
     NTINUE***";: GET A$: RETURN
```

After you have typed in the example on page 23, "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:

GR

COLOR =

PLOT

HOME

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

COLOR = command sets the next color to be plotted to the value of expression following "COLOR=". Color remains set until changed by a new "COLOR =" command. For example, the color plotted in line 160 remains the same no matter how many points are plotted. The value of the expression following "COLOR=" must be in the range of Ø to 15 or an error may occur.

Change the program by re-typing in lines 150 and 160 as follows:

```
15Ø INPUT "ENTER X, Y, COLOR"; X, Y, Z

16Ø COLOR = Z : PLOT 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.

"PLOT X, Y" command plots a small square of color defined by the last COLOR = command at the position specified by expressions X and Y. Remember, X and Y must each be a number in the range of Ø to 39.

"HOME"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: <u>To get from color graphics back to all text mode</u>, type"TEXT" and depress "RETURN" key <u>if you have the "]" prompt character.</u>

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

```
10  GR : HOME
20  FOR I = 0 TO 31
30  COLOR= I / 2
40  VLIN 0,39 AT I
50  NEXT
60  FOR I = 0 TO 14 STEP 2: PRINT
      TAB( I * 2);I;: NEXT
70  PRINT : FOR I = 1 TO 15 STEP
     2: PRINT  TAB( I * 2);I;: NEXT

80  PRINT : PRINT "STANDARD APPLE
      COLOR BARS";
```

Color bars are displayed at double their normal width. The left most bar is black as set by COLOR = Ø; the right most, white, is set by COLOR=15. Depending on the tint setting on your TV, the second bar as set by COLOR = 1 will be magenta (reddish-purple) and the third will be blue. Adjust your TV tint control for these colors. In Europe, color tints may be different.

In the last program a new command of the form VLIN Y1,Y2 AT X was used in line 4Ø. 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 HLIN X1, X2 AT Y is similar to VLIN except that it plots a horizontal line.

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

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 DEL, 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.
DEL	DEL X,Y	Deletes lines X to Y, inclusive, from the program. Note that both line numbers must be present. This statement may be used inside a program, but will stop program after "DEL" statement is executed.
HIMEM:	HIMEM: 16384 60 HIMEM: 2400	Sets last memory location available to BASIC program including variables. Used to protect area of memory for machine language routines or data. Statement may be used inside program.
LIST	LIST X	Lists line "X" if there is one. LIST Ø is not allowed.
	LIST or 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- or LIST X,	Lists all lines in a program with a line number equal to or greater than "X".
	LIST -X or LIST, X	Lists all of the lines in a program with a line number less than or equal to "X".
	LIST X-Y or 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.
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 2ØØ	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.
SPEED	SPEED=50 200 SPEED=225	Sets speed at which characters are outputted, either to TV screen or to other I/O devices. Ø is slowest speed; 255 is fastest.

Arithmetic Operators

SYMBOL	SAMPLE STATEMENT	PURPOSE/USE
=	A=1ØØ LET Z=2.5	Assigns a value to a variable. The LET is optional.
-	B=-A	Negation. Note that Ø-A is subtraction, while -A is negation.
↑ (↑ is a shift-n)	13Ø PRINT X↑3	Exponentiation (equal to X*X*X in the sample statement). Ø↑Ø=1; Ø to any other power = Ø;A↑B with A negative and B not an integer gives an "ILLEGAL QUANTITY" error.
*	14Ø X=R*(B*D)	Multiplication
/	15Ø PRINT X/1.3	Division
+	16Ø Z=R+T+Q	Addition
-	17Ø J=1ØØ-I	Subtraction

Logical and Relational Operators

		PURPOSE/USE
=	1Ø IF A=15 THEN 4Ø	Expression Equals Expression
<>	7Ø IF A<>Ø THEN 5	Expression Does Not Equal Expression
>	3Ø IF B>1ØØ THEN 8	Expression Greater Than Expression
<	16Ø IF B<2 THEN 1Ø	Expression Less Than Expression

Logical and Relational Operators (Cont.)

SYMBOL	SAMPLE STATEMENT	PURPOSE/USE
<=,=<	18Ø IF 1ØØ<=B+C THEN 1Ø	Expression Less Than Or Equal To Expression
>=,=>	19Ø IF Q=>R THEN 5Ø	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=Ø (NOT true=false)

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+1Ø/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) NEGATION — -X WHERE X MAY BE A FORMULA

3) NOT — LOGICAL "NOT" IS LIKE NEGATION. "NOT" TAKES ONLY THE FORMULA TO ITS RIGHT AS AN ARGUMENT.

4) ↑ — EXPONENTATION

5) * / — MULTIPLICATION AND DIVISION

6) + - — ADDITION AND SUBRACTION

7) RELATIONAL OPERATORS (equal precedence for all six)
   - = EQUAL
   - <> NOT EQUAL
   - < LESS THAN
   - > GREATER THAN
   - <= LESS THAN OR EQUAL TO
   - >= GREATER THAN OR EQUAL TO

8) AND — LOGICAL "AND"

9) OR — LOGICAL "OR"

Relational Operator expressions will always have a value of True (+1) or a value of False (Ø). Therefore, (5=4)=Ø, (5=5)=+1 (4>5)=Ø, (4<5)=+1, etc.

The THEN clause of an IF statement is executed whenever the formula after the IF is <u>not</u> equal to Ø. That is to say, IF X THEN...is equivalent to IF X<>Ø 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 I or J denotes an expresssion 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.Ø1 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
CALL	10 CALL 300 20 CALL X*Y 30 CALL - 936 40 CALL 64600	Causes execution of a machine level language subroutine at decimal memory location specified. Locations above +32767 may also be specified as a negative number below 65636; i.e., lines 30 and 40 are identical.
COLOR=	COLOR=I	Sets TV display color to value in expression I. Expression I must be in the range of Ø to 15. Colors are assigned the values: Ø - Black, 1 - Magenta, 2 - Dark Blue, 3 - Light Greeen, 4 - Dark Green, 5 - Grey, 6 - Medium Blue, 7 - Light Blue, 8 - Brown, 9 - Orange, 10 - Grey, 11 - Pink, 12 - Green, 13 - Yellow, 14 - Blue/Green, 15 - White NOTE: Color may vary on European (625 line) T.V. Color remains set until a new "COLOR=" command changes it or until a "GR" command clears screen and sets COLOR=Ø.
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 restricted 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.
	11Ø 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 88 dimensions are allowed but in practice 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.
DRAW	140 DRAW S AT X,Y	Draws a HIRES shape starting at the coordinates specified by expressions X and Y. The shape drawn is specified by expression S whose description is in the shape table previously loaded using "SHLOAD" command. The color, rotation and scale of shape draw must have been previously specified.
	150 DRAW S	Same as above but draws a shape as specified by expressions starting at last point plotted by previous HPLOT, DRAW, or XDRAW command.
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=10*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.
FLASH	5Ø FLASH	Sets video mode for output characters to "Flashing"; ie. alternating between, normal and inverse.
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.
GR	53Ø GR	Switches TV screen display from all text mode into color graphics (40x40) with 4 lines of text at bottom of screen
	55Ø GR:POKE -163Ø2,Ø	Sets all color graphics (40x48) mode with no text at bottom.
HCOLOR=	7Ø HCOLOR=I	Sets high resolution line color to that specified by expression following "HCOLOR=" which must be in the range of Ø to 7; where: 0 = Black 1 1 = Green 2 = Blue 3 = White 1 4 = Black 2 5 = (depends of TV) 6 = (depends on TV) 7 = White 2
HIMEM:	HIMEM: 16384 6Ø HIMEM: 2400	Sets last memory location available to BASIC program including variables. Used to protect area of memory for machine language routines or data. Statement may be used inside program.
HGR	1Ø HGR 2Ø HGR: POKE -163Ø2,Ø	Sets mixed screen high resolution graphics video mode (280x160+4 lines of text) and displays page 1 of memory (8k-16k) and clears screen to black. Text screen memory is not affected. NOTE: This command cannot be used with the cassette version of APPLESOFT II because the APPLESOFT language resides in the same memory space as the screen refresh information in the HGR mode. Example 20 sets all high resolution graphics mode (280x192 with no text at bottom of screen.)
HGR2	3Ø HGR2 4Ø HGR2: POKE -163Ø1,Ø	Sets all screen high resolution graphics video mode (280x192) and displays page 2 of memory (16k-24k) and clears screen to black. Example 40 sets mixed HIRES mode on page 2 and is not allowed.
	5Ø POKE -163Ø4,Ø: POKE -163Ø2,Ø: POKE -16299,Ø: POKE-16297	Sets all screen HIRES mode page 2 without clearing screen to black. (See special controls)
HLIN	HLIN X1,X2 AT Y	If in color graphics mode, this command draws a horizontal line, of color as set by "COLOR=", from coordinate X1 to X2 at position Y. Numeric value for X1, X2 and Y must be between Ø and 39. (Y may range up to 47 if in all color modes; i.e., no 4 lines of text at bottom of screen.)
	HLIN Ø,19 AT Ø	Draws horizontal line along the top of the TV screen from upper-left corner to center of screen.
	HLIN 2Ø, 39 AT 39	Draws horizontal line along the bottom of the TV screen from bottom center to lower-right corner.
HOME	7Ø HOME	Moves cursor to upper left screen position within scrolling window and clears all text within the window. See special controls and features section of Applesoft manual on how to set scrolling window.
HPLOT	8Ø HPLOT X,Y	Plots a HIRES point in color specified by previous "H COLOR=" command at the position specified by expressions X and Y.
	90 HPLOT X1,Y1 TO X2,Y2	Draws a HIRES line in color specified by previous "H COLOR=" command from coordinates specified by expressions X1 and Y1 TO the coordinate specified by expressions X2 and Y2.
	100 HPLOT TO X2,Y2	Draws a line from last position plotted to coordinates specified by expressions X2 and Y2. NOTE: HCOLOR may not be changed when using this command.

NAME	EXAMPLE	PURPOSE/USE
HTAB	8Ø HTAB 1Ø	Moves cursor to absolute horizontal position independent of current cursor postion.
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 if 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.
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.
INVERSE	13Ø INVERSE	Sets video mode for output characters to inverse; i.e., black letters on white background.
IN#	1ØØ IN# 6 11Ø IN# Y+2 12Ø IN# Ø	Transfers source of data for subsequent "INPUT" statements to peripheral I/O slot (1-7) specified. Slot Ø is not addressable from BASIC. IN# Ø (Example 12Ø) is used to return data source from peripheral I/O to keyboard. If no Apple peripheral is in slot specified, the system will hang up. To recover, hit "RESET" key then type "ØG" and depress "RETURN" key.
LET	3ØØ LET W=X	Assigns a value to a variable and is optional.
	31Ø V=5.1	
LOMEM:	15Ø LOMEM: 16384	Sets starting memory location of first BASIC variable. Normally "LOMEM:" is set automatically to the end of current program by Applesoft. This command is added to allow protection of variables from High Resolution Graphics in large memory size systems. Must be used inside program. Once program is modified, LOMEM: is automatically reset.
NEXT	34Ø 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 34Ø.
	35Ø NEXT V,W	A single NEXT may be used to match miltiple FOR statements. Equivalent to NEXT V:NEXT W.
NORMAL	16Ø NORMAL	Sets video mode for output characters to normal; i.e., white letters on black background.
NO TRACE	NO TRACE 17Ø NO TRACE	Turns off "TRACE" debug mode described below.
ONERRGOTO	1Ø ONERRGOTO 1ØØØ	Sets a flag that causes unconditional jump (later in the program) to program line number specified by expression X when an error condition occurs instead of printing error message and halting program execution.
ON...GOTO	1ØØ ON I GOTO 1Ø,2Ø,3Ø,4Ø	Branches to the line indicated by the I'th number after the GOTO. That is: IF I=1, THEN GOTO LINE 1Ø IF I=2, THEN GOTO LINE 2Ø IF I=3, THEN GOTO LINE 3Ø IF I=4, THEN GOTO LINE 4Ø If I<1 or I attempts to select a nonexistent line (I>4) in this case, the statement after the ON statement is executed. However, if I is >255 or <Ø, an "ILLEGAL QUANTITY" error message will result. As many line numbers as will fit on a line can follow an ON...GOTO.
	1Ø5 ON SGN (X) +2 GOTO 4Ø,5Ø,6Ø	This statement will branch to line 4Ø if the expression X is less than zero, to line 5Ø if it equals zero, and to line 6Ø if it is greater than zero.

STATEMENTS (CONT)

NAME	EXAMPLE	PURPOSE/USE
ON...GOSUB	11Ø ON I GOSUB 5Ø,6Ø	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.
PLOT	63Ø PLOT X,Y	Plots a small square of color set by "COLOR=" 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 PLOT 2Ø,2Ø	Plots a small square at center of screen.
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.
POP	18Ø POP	"POPS" Nested "GOSUB" return stack address by one.
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.
PR#	19Ø PR#7	Like IN#, transfers output to I/O slot defined by expression after "PR#". PR# Ø returns output to video port and not to slot #Ø.
READ	49Ø READ V,W	Reads data into specified variables from a DATA statement. The first piece of data read will be the first piece of data listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA STATEMENT, and so on. When all of the data have been read from the first DATA statement, the next piece of data to be read will be the first piece listed in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an "OUT OF DATA" error. The line number given in the "SYNTAX ERROR" will refer to the line number where the error actually is located.
RECALL	2ØØ RECALL A 21Ø RECALL A%	Reads into matrix A the data from cassette tape previously saved using the "STORE" command. Array names are not stored along with their values so that an array may read back using the "RECALL" command with a different matrix variable name than the one used with the "STORE" command. When "RECALL"ing an array, the size must be identical to the original array or the first index only may be larger. For example if A(7,10) is stored, then one may recall A(7,10) or A(20,10) but not A(7,20).
REM	5ØØ REM NOW SET V=Ø	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=Ø will be executed.
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.
RESUME	1ØØØ RESUME	Causes resumption of program at the point where an error occured. If "RESUME" is encountered before an error occurs, program will be klobbered. If error occurs in an error handling routine, the use of "RESUME" will place program in infinite loop and "RESET" key must be depressed in order to escape.
ROT =	12Ø ROT = W	Sets angular rotation for shape draw to value in range of Ø to 63 as specified by expression W. ROT = Ø is Ø degrees, ROT =16 is 90 derees, ROT =32 is 180 degrees, etc. For SCALE=1 only 4 rotation values are allowed (0,16,32,48); for SCALE =2,8 rotations; etc.
RETURN	7ØØ RETURN	Causes a subroutine to return to the statement after the most recently executed GOSUB.
SCALE =	11Ø SCALE =Z	Sets scale size for shape drawing to factor from 1 to 255 as specified by expression Z. NOTE: SCALE = Ø is maximum size and not a single point.

STATEMENTS (CONT.)

NAME	EXAMPLE	PURPOSE/USE
SHLOAD	13Ø SHLOAD	Loads a shape table from cassette tape. Table is loaded from HIMEM: down and HIMEM: is set to just below the shape table to protect it. If a second shape table is loaded, HIMEM: should be reset to avoid wasting memory. Shape table tapes are prepared using programs supplied on High Resolution Shapes cassette tape (Apple P/N A2T0005X)
SPEED=	SPEED=5Ø 2ØØ SPEED=255	Sets speed at which characters are outputted, either to TV screen or to other I/O devices. Ø is slowest speed; 255 is fastest.
STOP	9ØØØ STOP	Causes a program to stop execution and to enter command mode. Prints BREAK IN LINE 9ØØØ (as per this example). CONT after a STOP branches to the statement following the STOP.
STORE	23Ø STORE A 23Ø STORE A% 24Ø STORE A$	Saves the data in array A onto cassette tape that is read back into memory with a "RECALL" command. Only floating point or integer arrays may be "STORE"d. String arrays (Ex 24Ø) may not be "STORE"d. In order to save string array data; it must be first converted to a numerical array using the "ASC" function.
TEXT	8ØØ TEXT	Sets TV display to all text mode from color graphics mode and resets TV display to 24 lines of 4Ø characters each if otherwise. Returns to text mode from GR or HGR. Sets scrolling window to maximum. HIRES screen memory is not affected.
TRACE	TRACE 21Ø TRACE 22Ø IF X<Ø THEN TRACE	Sets a debug mode that displays the line number of each statement as it is executed.
VTAB	23Ø VTAB 18 24Ø VTAB Z+2	Moves cursor to absolute verticaL postion as specified by expression after "VTAB". VTAB 1 is top line. VTAB 24 is bottom line.

INTRINSIC FUNCTIONS

NAME	EXAMPLE	PURPOSE/USE
ABS (X)	12Ø PRINT ABS (X)	Gives the absolute value of the expression X. ABS returns 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 -π/2 to π/2. (π/2=1.57Ø8)
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.
FRE(X)	16Ø PRINT FRE (Ø)	Gives the number of memory bytes currently unused by BASIC.
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*10↑D+.5)/INT(10↑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 (I)	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 garbage. (see POKE statement)
PDL(X)	35Ø PRINT PDL(X)	Gives number between Ø and 255 corresponding to paddle position on game paddle number designated by expression (X) and must be legal paddle number (0,1,2,or 3).
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.

INTRINSIC FUNCTIONS (CONT.)

NAME	EXAMPLE	PURPOSE/USE
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 generates a new random number between Ø and 1. Note that V-A(*RND(L)+A will generate a random number between A & B.
SCRN(X)	38Ø PRINT SCRN(X1,Y1)	Gives color (number between Ø and 15) of screen at horizontal location designated by expression X1 and vertical location designated by expression Y1 Range of express X1 is Ø to 39. Range of expression Y1 is Ø to 39 if in standard mixed colorgraphics display mode as set by GR command or Ø to 47 if in all color mode set by POKE -16304 ,Ø: POKE -163Ø2,Ø.
SGN(X)	22Ø PRINT SGN(X)	Gives 1 if X>Ø, Ø if X=Ø and -1 if X<Ø.
SIN(X)	230 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 and will not back-up cursor. See HTAB command.
TAN(X)	26Ø PRINT TAN(X)	Gives the tangent of the expression X. X is interpreted as being in radians.

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	40 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 or a "STRING TOO LONG" error will occur.
PRINT	6Ø PRINT X$ 7Ø PRINT "FOO"+A$	Prints the string expression on the screen.
READ	50 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.
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 NOTdisplayed on the TV screen. B^C and C^C must be followed by a carriage return. Screen editing characters areindicated 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". If you change any data in the range of $0,1FF while in the monitor, you will have to re-load Applesoft.
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
U^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. "GR" 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. "TEXT" 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 "GR" 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 will 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.

SPECIAL CONTROLS AND FEATURES (CONT.)

BASIC Examples	DESCRIPTION
170 CALL-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 CALL-868	Clear current line from cursor to right margin. This is the same as E_E (Escape E).
190 CALL-922	Issues a line feed.
200 CALL-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.
220 X=PEEK(-16336)	Toggle speaker once.
230 X=PEEK(-16384)	Read keyboard; if X>127 then key was depressed and X is ASC II value of key depressed with bit 7 set. 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=PEEK(-16368,Ø)	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 POKE-16296,1	Set Game I/O output #Ø to TTL high (3.5 volts).
290 POKE-16295,0	Set Game I/O output #0 to TTL low (0.3 volts).
300 POKE-16294,1	Set Game I/O output #1 to TTL high (3.5 volts).
310 POKE-16293,Ø	Set Game I/O output #1 to TTL low (0.3 volts).
180 CALL 11246	(Cassette tape version) Clears current HIRES screen to black
190 HCOLOR=I: HPLOT Ø,Ø:CALL 11250	(Cassette tape version) Sets entire background to color specified by expression I.
200 CALL 11719: HPOLT TO X2,Y2	Draws a line from end of last shape drawn to point X2,Y2
210 CALL 11719: Y= PEEK (226):X=PEEK (224) + PEEK (225)*256	Finds X and Y coordinates for the end of last shape plotted.
1010 X=PEEK (218)+ PEEK (219) *256	This statement sets X equal to the line number of the statement where an error occured if an ONERRGOTO statement has been executed..
1020 IF PEEK (216)>127 THEN GOTO 2000	If Bit 7 at memory (ERRFLG) location 222 has been set true, then an "ONERRGOTO" statement has been encountered.
1030 POKE 216,0	Clears ERRFLG so that normal error messages will occur
1040 Y=PEEK (222)	Sets variable Y to a code that described type of error that caused an "ONERRGOTO" jump to occur. Error types are described below:

Y VALUE	ERROR TYPE ENCOUNTERED	Y VALUE	ERROR TYPE ENCOUNTERED
0	Next without for	133	Division by Zero
16	Syntax	163	Type Mismatch
22	Return without Gosub	176	String too long
42	Out of Data	191	Formula too complex
53	ILLEGAL QUANTITY	224	Undefined Function
69	Overflow	254	Bad response to an input statement
77	Out of memory		
90	Undefined Statement	255	Control-C interrupt attempted
107	Bad Subscript		
120	Redimensioned Array		

APPENDICES

APPENDIX A

Getting APPLESOFT 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) or you will need the Applesoft ROM BASIC peripheral card (Apple Part No. A2B0009X) The cassette tape version of APPLESOFT BASIC occupies approximately 10k bytes of memory, thus a computer with 16k bytes or more memory is required to use APPLESOFT BASIC. A 4k minimum system is required with the APPLESOFT ROM card.

Cassette version of 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 transfer to Applesoft language.

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.

Ø3Ø-ØØ15

HOW TO INSTALL AND USE THE APPLESOFT II FIRMWARE CARD

INSTALLATION

To install the APPLESOFT card you will simply plug it into a socket inside the APPLE II. Care must be exercised, however, and these instructions should be followed exactly:

1. Turn the APPLE II off. This is <u>very</u> important to prevent damage to the computer.

2. Remove the cover from the APPLE II. This is done by pulling up on the cover at the rear edge (the edge farthest from the keyboard) until the two corner fasteners pop apart. Do not continue to lift the rear edge, but slide the cover backward until it comes free.

3. Inside the APPLE II, across the rear of the circuit board, there is a row of eight long, narrow sockets called "slots". The leftmost one (looking at the computer from the keyboard end) is slot #Ø, and the rightmost one is slot #7. Holding the APPLESOFT card so that its switch is toward the rear of the computer, insert the "fingers" portion of the card into slot #Ø, the leftmost one. The "fingers" portion will enter the socket with some friction and will then seat firmly. The APPLESOFT card must be placed in slot Ø.

4. The switch on the back of the APPLESOFT card should protrude part way through the slot on the back of the APPLE II.

5. Replace the cover of the APPLE II, remembering to start by sliding the front edge of the cover into place. Press down on the two rear corners until they pop into place.

7. The APPLESOFT card is now installed, and the APPLE II may be turned on.

USING THE APPLESOFT CARD

With the APPLESOFT card's switch in the downward position, the APPLE II will begin operating in Integer BASIC when you use {RESET}{CTRL}B. With the switch in the upward position, {RESET}{CTRL}B will bring up APPLESOFT BASIC instead of Integer BASIC.

When using the Disk Operating System, the computer will automatically choose Integer BASIC, or APPLESOFT from the card, as required. It does not matter in which position the switch is set.

To change from Integer BASIC to APPLESOFT, or vice-versa, without operating the switch, the following commands may be used:

{RESET}CØ8Ø{RETURN}
{CTRL}B{RETURN}
will put the computer into APPLESOFT, and

{RESET}CØ81{RETURN}
{CTRL}B{RETURN}
will put the computer into Integer BASIC.

CORRECTING THE APPLESOFT ON CARD AND APPLESOFT ON DISK INCOMPATIBILITY

Application note: 24 JULY 78

If a program was generated using the version of APPLESOFT that is on the disk, it will no longer run once the APPLESOFT card has been installed. It is very easy to convert the program so that it will run.

1. LOAD the program, but do not RUN it.

2. Type the command
CALL 54514

3. SAVE the program. You may use the same name if the original file is UNLOCKed.

If a program was generated using the version of APPLESOFT from the card, it will no longer run in an APPLE II that doesn't have the APPLESOFT card. It is possible to convert the program so that it will run from the version of APPLESOFT that resides on the disk.

1. LOAD the program, but do not RUN it.

2. Type the command
CALL 3314

3. SAVE the program. You may use the same name if the original file is UNLOCKed.

APPENDIX B
Program Editing with APPLESOFT BASIC

Most ordinary humans make mistakes occasionally....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. 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 and an Escape B.

```
]LIST

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

CURSOR

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:

CURSOR

```
]LIST

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

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:

CURSOR

```
]LIST

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

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 1Ø.

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 1Ø. 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 D_E 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:

CURSOR

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

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

CURSOR

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

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

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

CURSOR

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

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

CURSOR

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 output 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, DEF, or DATA 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) SQR-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, POKE, TAB, SPC , ON.. GOTO, or any of the graphics functions 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 (5 bytes) 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. (A single line can include up to 254 characters.)

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

3) 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.

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

```
10 P=3.14159
```

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

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

6) Re-use the same variables. If you have a variable T which is used to hold a temporary result in one part of the program and you need a temporary variable later in your program, use it again. Or, if you are asking the

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.

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

8) Use the zero elements of matrices; for instance, A(Ø), B(Ø,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 1Ø.

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=Ø: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) During program execution, when APPLESOFT encounters a new line reference such as "GO TO 1ØØØ" it scans the entire user program starting at the lowest line until it finds the referenced line number (1ØØØ in this example). Therefore, frequently referenced lines should be placed as early in the program as possible.

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)
COTANGENT	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: 5ØØ LET B=C=Ø. 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 Ø. If C did not equal Ø, B would be set to Ø. 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.

APPLESOFT CONVERT PROGRAM

Programs written for APPLESOFT and saved on tape cannot be LOADed and RUN with APPLESOFT II. Instead of retyping these programs, you can use the CONVERT program, which runs in INTEGER BASIC. This program accepts a tape in APPLESOFT BASIC and produces a new tape in APPLESOFT II BASIC.

TO USE IT:

LOAD THE CONVERT TAPE. It will ask you if the old program (the one in APPLESOFT BASIC) used in OPTION 1 or OPTION 2.

OPTION 1 was GRAPHICS COMMANDS WITHOUT LET OR REM STATEMENTS

OPTION 2 was LET OR REM STATEMENTS, BUT NO GRAPHICS

After you answer, you will be prompted to play the old program tape. After the program has finished reading and processing your old tape, you will be asked to record a second tape. The second tape will be your program converted into APPLESOFT II.

If any errors are discovered, self-explanatory messages are given.

FOLLOWING IS THE LISTING OF THE CONVERTFROM APPLESOFT I TO APPLESOFT II PROGRAM. THIS PROGRAM IS WRITTEN ININTEGER BASIC, AND CAN BE RUN ON ANY SYSTEM LARGE ENOUGH TO RUN APPLESOFT

```
    0 TEXT : CALL -936: VTAB 3: PRINT "APPLESOFT CONVERSION PROGRAM:"

    2 PRINT "CONVERTS OLD APPLESOFT PROGRAMS TO": PRINT "APPLESOFT ][ F
      ORMAT"
    3 PRINT "COPYRIGHT 1978 APPLE COMPUTER, INC.": PRINT
    4 PRINT : PRINT : PRINT : POKE 34,10
    5 PRINT "WAS PROGRAM WRITTEN IN OPTION 1 OR": PRINT "OPTION 2?"
      : PRINT "OPTION 1: GRAPHICS COMMANDS WITHOUT"
    6 PRINT "                LET OR REM STATEMENTS": PRINT "OPTION 2: LET AN
      D REM STATEMENTS BUT NO              GRAPHICS."
    7 INPUT "OPTION #",O: IF O<>1 AND O<>2 THEN 7
   10 CALL -936: PRINT "PUT APPLESOFT PROGRAM TAPE IN RECORDER,": POKE
      60,Z: POKE 61,Z: POKE 62,2: POKE 63,Z:F=1536:B=4096
   20 INPUT "PRESS THE PLAY BUTTON, THEN HIT RETURN",A$: CALL -259

   25 IF PEEK (1)<128 THEN 30: PRINT "TAPE READ ERROR!!": PRINT "TRY RE
      -ADJUSTING VOLUME CONTROLS ON TAPEPLAYER, THEN RE-RUN THIS PROGRA
      M": END
   30 POKE 60,Z: POKE 61,16:E= PEEK (Z)+ PEEK (1)*256-6657: POKE 62
      ,E MOD 256: POKE 63,E/256: CALL -259
   35 CALL -936: PRINT : PRINT : PRINT "CONVERTING..."
   40 PRINT : PRINT : PRINT : POKE 34,10
   50 FOR B=B+4 TO B+999:T= PEEK (B): IF T<133 THEN 250: IF T<>135
       AND T<>142 OR O=2 THEN 200:C=B
   55 IF T<>142 THEN 60:T=137: GOTO 250
   60 C=C+1:U= PEEK (C): IF U=32 THEN 60: IF U=67 OR U=71 OR U=72 OR
      U=80 OR U=86 THEN GOTO U: PRINT "BAD STATEMENT IN PROGRAM": GOTO
      250
   67 T=160: GOTO 90
   71 T=136: GOTO 90
   72 T=142: GOTO 87
   80 T=141: GOTO 90
   86 T=143
   87 CC=Z:D=B
   88 D=D+1: IF PEEK (D)<>44 AND PEEK (D)<>58 AND PEEK (D) THEN 88
      : IF PEEK (D)=44 THEN 89: PRINT "BAD STATEMENT IN PROGRAM!":
       GOTO 250
   89 CC=CC+1: IF CC=1 THEN 88: POKE D,197
   90 POKE C,32: GOTO 250
  199 REM :MAP OLD TOKENS TO NEW
  200 IF T>195 THEN 250:T=T+1+(T>134)*34+(T>139)+(T>160)+(T>177)*2

  250 POKE B,T: IF B/500*500=B THEN PRINT "STILL CONVERTING!"
  251 IF T<>0 THEN NEXT B:B=B+1: GOTO 40
  878 CC=Z:D=C
 1000 CALL -936: POKE 60,Z: POKE 61,Z: POKE 62,2: POKE 63,Z: PRINT
      "DONE!
": INPUT "START RECORDING, THEN HIT 'RETURN'",A$
 1001 POKE E-2,Z: POKE E-1,Z: POKE E,Z
 1005 D=E-4096: POKE Z,D MOD 256: POKE 1,D/256: POKE 2,Z: CALL -307

 1010 POKE 60,Z: POKE 61,16: POKE 62,E MOD 256: POKE 63,E/256: CALL
      -307
 1020 PRINT "O.K.": PRINT "THE TAPE JUST RECORDED CAN NOW BE LOADED INT
      O APPLESOFT ][.": END
```

APPENDIX H

ASCII Character Codes

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
0	NULL	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	&	70	F
7	BEL	39		71	G
8	BS	40	(	72	H
9	HT	41	)	73	I
10	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	0	80	P
17	DC1	49	1	81	Q
18	DC2	50	2	82	R
19	DC3	51	3	83	S
20	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	:	90	Z
27	ESCAPE	59	;	91	[
28	FS	60	<	92	\
29	GS	61	=	93	]
30	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.
3ØØ.3FF	Available to user for short machine language programs.
4ØØ.7FF	Screen display area for text or color graphics.
8ØØ. 2FFF	APPLESOFT BASIC compiler. (Cassette Tape Version)
8ØØ.XXX	User Program (ROM version - A2B0009X installed) where XXX is maximum available RAM memory
2000.3FFF	High Resolution Graphics Display page 1. May be used by ROM (A2B0009X) version of Applesoft II only.
3000.XXX	User program (Cassette Tape Version) 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.
4000.5FFF	High resolution graphics display page 2.
CØØØ.CFFF	Hardware I/O Addresses.
DØØØ.DFFF	Future ROM expansion
DØØØ.F7FF	Applesoft II ROM version with select switch "ON".
EØØØ.F7FF	Apple Integer BASIC
F8ØØ.FFFF	Apple System Monitor

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.

Arbib, 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.

Applesoft
Zero Page Usage

LOCATION(s) (in hex)	USE
Ø-5	Jump instructions to continue in Applesoft. (ØG for Applesoft is equivalent to Control-C for integer Basic)
$A-$C	Location for USR() function jump instruction. See USR() function description.
$D-$17	General purpose counters/flags for Applesoft.
$20-$4F	Apple II system monitor reserved locations.
$50-$61	General purpose pointers for Applesoft.
$62-$66	Result of last multiply/divide.
$67-$68	Pointer to beginning of program. Normally set to $Ø8Ø1 for ROM version, or $3ØØ1 for RAM (cassette tape) version.
$69-$6A	Pointer to start of simple variable space. Also points to the end of the program plus 5, unless manually changed with the LOMEM: statement.
$6B-$6C	Pointer to beginning of Array space.
$6D-$6E	Pointer to end of numeric storage in use.
$6F-$70	Pointer to start of string storage. Strings are stored from here to the end of memory.
$71-$72	General pointer.
$73-$74	Highest location in memory available to Applesoft plus one. Upon initial entry to Applesoft, is set to the end of memory available.
$75-$76	Current line number of line being executed.
$77-$78	'Old line number'. Set up by a control-C, STOP or END statement. Gives line number that execution was interrupted at.
$79-$7A	'Old text pointer'. Points to location in memory forstatement to be executed next.
$7B-$7C	Current line number where DATA is being read from.
$7D-$7E	Points to absolute location in memory where DATA is being read from.
$7F-$80	Pointer to where input is com_ing from currently. Is set to $2Ø1 during an INPUT statement, or

LOCATION(S)	USE
	during a READ statement is set to the DATA in the program it is READing from.
$81-$82	Holds the last used variable name.
$83-$84	Pointer to the last used variable's value.
$85-$9C	General usage.
$9D-$A3	Main floating point accumulator.
$A4	General use in floating point math routines.
$A5-$AB	Secondary floating point accumulator.
$AC-$BØ	General usage flags/pointers.
$B1-$C8	CHRGET routine. Applesoft calls here everytime it wants another character.
$B8-$B9	Pointer to last character obtained through the CHRGET routine.
$C9-$CD	Random number.
$DØ-$D5	High resolution graphics scratch pointers.
$D8-$DF	ONERR pointers/scratch.
$EØ-$E2	High-resolution graphics X and Y coordinates.
$E4	High-resolution graphics color byte.
$E5-$E7	General use for high resolution graphics.
$E8-$E9	Pointer to beginning of shape table.
$EA	Collision counter for high-resolution graphics.
$FØ-$F3	General use flags.
$f4-$F8	ONERR pointers.

apple computer inc.

10260 BANDLEY DRIVE
CUPERTINO, CALIFORNIA 95014 U.S.A.
TELEPHONE (408) 996-1010