PRELIMINARY

APPLE BASIC USERS MANUAL

OCTOBER 1976

Apple Computer Company • 770 Welch Rd., Palo Alto, CA 94304 • (415) 326-4248

This is a PRELIMINARY manual. It will, most likley, contain errors, incorrect wordings, etc. Your effort in noting these areas of improvement will be greatly appreciated.
If you find an error or can suggest an improvement, please write:

APPLE COMPUTER COMPANY
770 WELCH RD. SUITE 154
PALO ALTO, CA 94304
(415) 326-4248

LOADING APPLE BASIC

Apple BASIC is provided on a cassette tape which can be read into the "E" block of memory in about 30 seconds. To load BASIC from the tape do the following:

Hit the "CLEAR SCREEN" button which will clear the screen and display only the cursor (a flashing @ sign) in the upper left corner of the screen.

Hit the "RESET" button which will cause the computer to print a backslash (\) and move the cursor down a line.

Place the BASIC cassette into the recorder, rewinding it if necessary.

NOTE*** The symbol ↓ means hit the "RETURN" key on the keyboard. The symbol ↓ will not, of course, be printed on the screen.

1. Type C1ØØR ↓
2. Type EØØØ.EFFFR (don't hit ↓ yet!)
3. Start the tape
4. Hit ↓
5. When BASIC is loaded (about 30 seconds) the computer will print a backslash (\).

To enter BASIC type EØØØR↓. The computer will print a few characters, then, on the next line, print the prompt character >. This prompt character (>) is used throughout BASIC to signify that the BASIC is ready for additional commands or statements.

To exit BASIC hit the "RESET" button. This will return control to the monitor. To re-enter BASIC from the monitor without losing the previous program, enter BASIC at E2B3↓, instead of EØØØ↓. This is extremely useful when you have unintentionally hit the "RESET" button while in BASIC. Normally, you should enter BASIC at EØØØ↓, which clears any previous programs.

It is advisable to make a duplicate copy of Apple BASIC on another cassette. Exit BASIC as described and:

1. Type C1ØØR↓
2. Type EØØØ.EFFFW (don't hit ↓ yet!)
3. Start the tape (Recording)
4. Hit ↓.
5. The computer will print a backslash (\) when finished.

READING AND WRITING BASIC PROGRAMS ON TAPE

It is possible to store and retrieve BASIC programs on/from a cassette tape. To write a BASIC program onto a tape:

1. Exit BASIC as described above.
2. Type C1ØØR↓
3. Type 4A.ØØFFW8ØØ.FFFW (don't hit ↓ yet!)
4. Start the tape (Recording)
5. Hit ↓.
6. The computer will print a backslash (\) when completed.
7. Enter BASIC at E2B3.

To read a BASIC program from a tape, the same procedure is used except an R (for READ) is substituted for each W in line 3 above and the tape unit is playing instead of recording. Loading a BASIC program in this manner can be done either prior to loading BASIC or any time thereafter by first exiting BASIC, loading the program, and re-entering BASIC at E2B3.

PROGRAM EXECUTION

To review all of the program statements, the LIST command is used. To execute the program the RUN command is used, which causes the current program to be executed. Program execution may be interrupted by pressing any key. BASIC will then output a "STOPPED AT X" message to identify the point of interruption, where X is a line number.

NOTE*** A BASIC program can only be interrupted at the conclusion of a line. Therefore, the program:

```
10 FOR I=1 to 10 STEPØ: NEXTI
```

cannot be interrupted. It is good practice to separate potentially erroneous statements onto different lines to allow interruption if necessary.

The user may wish to examine or modify some of the program variables before resuming execution of the program. This can be done with BASIC commands, which execute immediately. For example, after interrupting a program, the commands:

PRINT A,B,C,D	Will print the values of A,B,C and D
A = 1ØØ	Will assign A = 1ØØ
PRINT A$	Will print the string A$

To resume execution of a BASIC program after interruption, type:

GOTO X , where X is the line number in the message "STOPPED AT X". GOTO X will begin execution at line number X without re-initializing all variables and strings in contrast to RUN, which re-initializes everything. Therefore, you must use GOTO X when resuming execution of the program.

ABBREVIATIONS

The following abbreviations are used in this manual:

expr	stands for an arithmetic expression.
var	is a variable name (numeric, array, or string).
val	is a number between -32767 and 32767 inclusive.
↓	indicates the pressing of the RETURN key.

NUMERIC REPRESENTATION

Apple BASIC can represent integers in the range -32767 to +32767. Entered values or calculations which result in values outside these boundaries will produce the error message ">32767".

VARIABLES

In Apple BASIC the allowed variables and variable names are:

Variable	Name	Example
numeric simple variable	letter or letter + digit	A, N A1, T6
numeric array variable	letter or letter + digit	N, T B1, T4
character string variable	letter + $	A$, N$

The same letter may be used to name any or all of the above types of variables in one program.

NOTE*** In Apple BASIC the first element of an array, A(1) is identical to the simple variable A.

EXPRESSIONS

An expression is a combination of numbers, variables, functions, and operators that can be, by calculation, reduced to a single value. The simplest expression is a number. Another simple expression is a variable name. Simple expressions can be combined to make arbitrarily complex expressions. Any expression may be enclosed in parentheses. Operations inside a pair of parentheses will be performed before any operations outside the parentheses.

ARITHMETIC AND RELATIONAL OPERATORS

-expr	negative one (-1) times the value of the expr.
NOT expr	Ø if expr is non-zero, 1 if expr is zero.
expr * expr	the product of the two expressions.
expr / expr	the quotient, truncated to an integer.
expr + expr	the sum of the two expressions.
expr - expr	the difference of the two expressions.

Relational expressions evaluate to one (1) if the condition is met, zero (Ø) if the condition is not met.

expr = expr	1 if expressions are equal.
expr > expr	1 if first expr greater than second.
expr < expr	1 if first expr less than second.
expr >= expr	1 if first expr greater than or equal to second.
expr <= expr	1 if first expr less than or equal to second.
expr <> expr	1 if the expr are unequal.
expr # expr	1 if expr are unequal, same as <> .
expr AND expr	1 if neither expr equals zero.
expr OR expr	1 unless both expr equal zero.
expr MOD expr	remainder left after dividing first expr by second.

FUNCTIONS

ABS(expr)	has the value of the expr when expr is zero or positive, and has the value of (-1 * expr) when expr is negative.
SGN(expr)	Ø if expr is zero, 1 if expr is positive, -1 if expr negative.
PEEK(expr)	is the value (decimal- between Ø and 255 inclusive) of the memory location whose (decimal) address is equal to expr.
RND(expr)	if expr is positive - gives a random integer between Ø and (expr -1). if expr is negative - gives a random integer between Ø and (expr+1).
LEN(var$)	returns the value equal to the number of characters currently assigned to the string whose name is var$.

ARRAYS

An array is a set of variables (numbers) assigned to a common variable name. Each variable of the set is identified by the name of the array followed by a parenthesized subscript. For example: A(3) references the third variable (number) of the array A. Other examples are: A(15), D(100), E(X).
In Apple BASIC, the first element of an array, B(1), is identical to the simple variable B. A reference to element zero (Ø) or a negative reference is an error, and will generate the error message "RANGE ERR".
Declaring arrays is done using a DIM statement, which gives the name of the array and its DIMensions.

The DIMension of an array specifies the number of variables in an array (the maximum allowable subscript). For example:
DIM A(15), N(6) assigns 15 variables to the array A (A(1) through A(15)) and six variables to the array N (N(1) through N(6)). There is no limitation on the number of variables dimensioned for an array other than restrictions due to available memory. If memory limitations are exceeded, a "MEM FULL ERR" will result.
NOTE*** Array variables are not initialized to any value.

STRINGS

Apple BASIC provides the user with the capability to manipulate character strings. A string is a sequence of characters which may include letters, digits, spaces and special characters (except quotation marks). A string literal (constant) is a string enclosed within quotation marks. String literals are often used in PRINT and INPUT statements -

```
100 PRINT "THIS IS A STRING LITERAL"
200 INPUT "X=", X
```

The quotation marks are not printed with the string. BASIC also permits the use of string variables. String values are assigned to string variables using the LET (or Implied LET) and INPUT commands. Apple BASIC strings function according to the following rules:

1. String variable names must be of the form: letter$ (Z$).
2. A string is DIMensioned for a maximum length using the DIM statement of the form: DIM A$(20), B$(100) ...
   A string may be DIMensioned to have a maximum length of from 1 to 255 inclusive. If an attempt is made to DIMension a string outside this range, the error message "RANGE ERR" will result.
3. If it is not specified in a DIM statement, a string's maximum length is taken to be zero (empty).
4. A string may contain fewer characters or the same number of characters as its maximum length, but may never contain more characters than its maximum length. If an attempt is made to exceed this maximum length, the error message "STR OVFL ERR" (string overflow error) will result.

SUBSTRINGS

Program statements using string variables may also use portions of strings (substrings) by subscripting the string variable name.

Where no subscript is specified, the entire string is referenced. If one subscript is specified - A$(5) for example - the characters occupying the 5th (in this case) through the last position inclusive are referenced.
If two subscripts are specified - A$(2,6) for example - the characters occupying the positions 2 through 6 inclusive are referenced.
Any numeric expressions may be used as subscripts. A$(I,J) for example, references the characters occupying positions I through J inclusive, where I and J are evaluated to character positions in the string and I is less than or equal to J.

For example, assume that A$="ABCDEFG", then

```
PRINT A$      yields     ABCDEFG
PRINT A$(5)   yields     EFG
PRINT A$(2,6) yields     BCDEF
PRINT A$(1,1) yields     A
```

DESTINGATION STRINGS

A destingation string is a string variable into which a different (source) string is being copied. Part or all of the destination string may be replaced by part or all of the source string.
Rules---

1. The destination string (to the left of the "=" sign) must be large enough to hold the source string.
2. If no subscripts are specified (A$=B$) then the entire source string (B$) replaces the entire string in the destination variable (A$). (If the source string is shorter than the destination string, trailing blanks are appended as necessary).
3. If one subscript is specified for the destination string (A$(5) = B$) then the destination string, beginning with the

specified character (the 5th character in this case) is replaced with the source string.

4. Specifying two subscripts (A$(3,5)) for the destination string is not allowed in Apple BASIC.

5. Zero, one or two subscripts may be specified for the source string, following the rules listed for substrings.

LEN FUNCTION

The LEN function returns the value equal to the number of characters currently assigned to a string variable.
Its form is: LEN(X$).
The length function can be used to link strings together as follows: B$(LEN(B$)+1) = A$. This will assign the characters from source string A$ to sequential character positions immediately following the last character previously assigned to the destination string, B$. The LEN function may be used with any program statement or command which has an expression (expr) argument.

STRING IF STATEMENT

Strings may be used in the relational expression of an IF - THEN STATEMENT. The logical operators allowed in Apple BASIC for a string IF statement are = and # (equal and not-equal). The strings are compared character by character on the basis of the ASCII character value. String variables may be subscripted in an IF statement (IF A$(3,7) = B$(4,8) THEN ...). If characters in the same positions are identical but one string has more characters than the other, the strings are considered not-equal.

BASIC INSTRUCTIONS

There are two kinds of instructions in BASIC: Commands and Statements. Commands are executed immediately after a ↵, do not have line numbers, and are not part of a program. Statements are always preceded by line numbers and become part of a program. Statements are executed only during the execution of a program.

Several BASIC instructions can be used both as statements and as commands. When used as commands, they execute immediately and are not part of a BASIC program. Used in this manner they can be useful for immediately examining or modifying program variables during interruption of program execution. With this feature, the Apple computer is also a simple calculator able to perform mathematical calculations immediately, without the necessity of writing a program. An instruction used as both a statement and a command:

>10 PRINT A,B	This is a statement in a BASIC program. Upon encountering line 10, a BASIC program will print the values of variables A and B.
>PRINT A,B	This is a command. The current values of A and B will be printed immediately after a ↵.

COMMANDS

The following commands (control commands) are used to enter, examine, modify and run BASIC programs. In addition to the control commands several BASIC instructions which can be used as commands are denoted in the list of statements.

AUTO *val1*, *val2*	starts automatically supplying line numbers. *val1* specifies the first line number value and *val2* specifies the increment between successive line number values. If *val2* is omitted, it is assumed to be ten (10). A control D (hitting the control key and D simultaneously) will terminate AUTO.
CLR	sets all variables to zero, cancels any pending FORs or GOSUBs and undimensions any array and string variables.

DEL *val1, val2* — erases from the program all lines numbered from *val1* to *val2* inclusive. If *val2* is omitted, just one line (*val1*) is DELeted.

LIST *val1, val2* — displays all program statements on lines numbered from *val1* to *val2*. If *val2* is omitted, just line *val1* is displayed. If both *val1* and *val2* are omitted, the entire program is LISTed.

RUN *val1* — does a CLR then initiates program execution beginning at line *val1*. If *val1* is omitted, then program execution starts at the lowest numbered line.

SCR — SCRatches (DELetes) the entire program. Nothing is saved.

HIMEM = (expr) — sets the high memory boundary for user programs (in decimal). Initializes to 4096.

LOMEM = (expr) — sets the low memory boundary for user programs (in decimal). Initializes to 2048. Both HIMEM and LOMEM destroy any current user programs.

STATEMENTS

Those BASIC instructions that can also be used as commands are denoted with a "C" in the left margin.

C LET *var* = expr or *var* = expr (Implied LET)

LET evaluates expr and asigns the resultant value to *var*. Use of the word LET is optional. Variables may be of any type (string, array, numeric).

INPUT item

An item may be any kind of variable name (string, array, numeric). An INPUT statement may contain several items separated with commas, each of which must be supplied a value.
INPUT prints a question mark (?) and awaits the user to input a value for the variable. A message can be printed prior to the "?" by preceding the list of variables by a message (in quotation marks) followed by a comma.
Responses to a multi-variable INPUT statement must be separated, using either a comma or a ↓ between each response. If an INPUT statement contains one or more

string variables, the responses must be separated with a ↓ (commas not allowed).

```
Examples:   INPUT A
            INPUT A,B,C$,D(2)
            INPUT "ENTER A,A$,B(3)", A,A$,B(3)
```

C PRINT item(s)

The item may be any kind of variable name, an expression, or a message to be printed. A message must be enclosed in quotes.
Any number of items may be printed using one PRINT statement. The items must be separated by either semicolons (;) or commas (,). The semicolon indicates that the items are to be printed with no intervening space. The comma forces the item following it to be printed in the next available column position. For this purpose the screen is thought of as consisting of five columns each eight characters wide.
A semicolon at the end of a list of items indicates that the next PRINT statement to be executed will begin printing exactly where the present one stopped. A terminating comma is illegal.

```
Examples:   PRINT A,C$,D(2)
            PRINT "message";A;"message";
            PRINT "A$=";A$;"--END"
```

C TAB (expr)

Prints the number of spaces equal to the value of expr (Modulo 256).

```
Examples:   TAB 20:  PRINT "Hello"
            PRINT A;:TAB 20:PRINT B
```

FOR *var* = expr1 TO expr2 STEP expr3

NEXT *var*

The FOR and NEXT statements form a pair. The FOR statement sets a numeric variable (*var*) equal to the value of expr1. Execution proceeds until the statement NEXT *var* occurs. At that time if *var* exceeds the value of expr2 execution continues from the statement following the NEXT *var*. If *var* does not exceed the value of expr2 then the value of expr3 is added to *var* and execution proceeds from the statement following the FOR. If STEP expr3 is omitted from the FOR statement, then expr3 is assumed to be +1.

```
Examples:   FOR I= 1 TO 100
            FOR A= 100 TO 1 STEP -5
            NEXT I
            NEXT I,J
```

C IF (expr) THEN statement IF (expr) THEN line number

If the value of the expression is zero no further action is taken and execution continues with the next statement following the IF statement. When the value of the expression is one (1) the statement following THEN is executed. A line number may also follow a THEN (instead of a statement). This instruction can be used as a command only if the instructions following THEN are also commands.

```
Examples:   IF A=B THEN C=1
            IF (A=B AND C=D) THEN 50
            IF NOT (A>4) THEN END
```

C GOTO expr

GOTO branches to the line number which equals expr.

```
Examples:   GOTO 100
            GOTO A
            GOTO (A+B/2)
```

GOSUB expr

RETURN

GOSUB and RETURN form a pair. GOSUB branches to the line number which equals expr. RETURN causes a branch to the line following the most recently executed GOSUB. There may be several conditional RETURNs in a GOSUB loop.

```
Examples:   GOSUB 100
            GOSUB A
            RETURN
```

C DIM *var*1 (expr1), *var*2 (expr2) ...

DIMensions an array or string named *var*1 to the value of expr1, an array or string named *var*2 to the value of expr 2, and so on. String and numeric variables may be mixed. An array or string may only be DIMensioned once in a program.

```
Examples:   DIM A(100)
            DIM A$(20),B$(10),C(50)
```

C REM text

REMark lets the user insert comments in a program without affecting the execution of the program. The comments immediately follow the REM statement and are preserved literally (spaces and all). REMarks are printed when listing the program.

```
Example:    REM THIS IS A REMARK
```

END

END stops execution of the program.

C POKE expr1, expr2

Puts the value of expr2 (decimal- must be between zero and 255 inclusive) into the memory location (decimal) whose value is equal to expr1.

```
Examples:   POKE 4,64          stores 64 in location 4
            POKE -2048,55      stores 55 in location -2048
                               (-2048 = D000 (HEX))
```

C CALL expr

CALL does a JSR to the memory location whose address is equal to the value of the expr (decimal). This statement links BASIC with assembly language subroutines. An assembly language RTS (return from subroutine) will return control to the BASIC and execute the next statement.

```
Examples:   CALL 64
            CALL A
```

NOTE*** Apple BASIC allows putting several statements on one line number. Each statement must be separated using a colon (:).

APPLE BASIC ERROR MESSAGES

*** SYNTAX	Results from a syntactic or typing error.
*** >32767 ERR	A value entered or calculated was less than -32767 or greater than 32767.
*** >255 ERR	A value restricted to the range Ø to 255 was outside that range.
*** BAD BRANCH ERR	Results from an attempt to branch to a non-existant line number.
*** BAD RETURN ERR	Results from an attempt to execute more RETURNs than previously executed GOSUBs.
*** BAD NEXT ERR	Results from an attempt to execute a NEXT statement for which there was not a corresponding FOR statement.
*** >8 GOSUBS ERR	Results from more than 8 nested GOSUBs.
*** >8 FORS ERR	Results from more than 8 nested FOR loops.
*** END ERR	The last statement executed was not an END.
*** MEM FULL ERR	The memory needed for the program has exceeded the memory size allotted.
*** TOO LONG ERR	Results from too many nested parentheses.
*** DIM ERR	Results from an attempt to DIMension a string array which has been previously dimensioned.
*** RANGE ERR	An array or string subscript was larger than the DIMensioned value or smaller than 1.
*** STR OVFL ERR	The number of characters assigned to a string exceeded the DIMensioned value for that string.
*** STRING ERR	Results from an attempt to execute an illegal string operation.
RETYPE LINE	Results from illegal data being typed in response to an INPUT statement. This message also requests that the illegal item be retyped. A backslash results when more than 128 consecutive characters are entered without an intervening ↓.