Notice

Apple Computer, Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

Disclaimer of All Warranties and Liabilities

Apple Computer, Inc. makes no warranties, either express or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Apple Computer, Inc. software is sold or licensed "as is." The entire risk as to its quality and performance is with the buyer. Should the programs prove defective following their purchase, the buyer (and not Apple Computer, Inc., its distributor, or its retailer) assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Apple Computer, Inc. be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software, even if Apple Computer, Inc. has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This manual is copyrighted. All rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Apple Computer, Inc.

© 1982 by Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

The word Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the U.S.A and Canada.

This manual was written for Apple Computer, Inc., by
Scot Kamins
Technology Translated
San Francisco, California

Reorder Apple Product #A2L2005
Welcome to volume two of the *Applesoft Reference Manual*. This volume contains the appendices, the glossary, and the index to both volumes. The paging continues where volume one left off.
Appendices

A. Summary of Applesoft Statements and Functions 215
B. Syntax Definitions 235
C. ASCII Character Codes 241
D. Reserved Words 245
E. Error Messages 247
F. Peaks, Pokes, and Calls 253
   F.1 Screen Text 253
   F.2 Keyboard 258
   F.3 Graphics 258
   F.4 Miscellaneous Input and Output 262
   F.5 Error Handling 264
G. Hints for Program Efficiency 267
   G.1 Saving Space 267
   G.2 Saving Time 270
H. Implementation Details 273
   H.1 Apple IIe Memory Map 274
   H.2 Applesoft Memory Allocation 275
   H.3 Zero Page Usage 278
   H.4 Keyword Tokens 280
I. Display Formats for Numbers 283
J. On-Screen Editing and Cursor Control 287
K. 40/80-Column Display Differences 289
L. Comparison with Integer BASIC 291
   L.1 Differences between Statements 292
   L.2 Other Differences 293
   L.3 Converting BASIC Programs to Applesoft 295
M. If You Have a Cassette Recorder 297
N. Complete Listing of the Postage Rates Program 301
Glossary 309
Index 331
Listed below are abbreviated descriptions of all Applesoft statements and functions. Each description is preceded by a syntactic definition and at least one example; see Appendix B for definitions of syntactic terms used here. References in square brackets at the end of each description give the section or appendix of this manual where more detailed information about the feature can be found.

**ABS**

*Syntax:* `ABS(aexpr)`  
*Example:* `ABS(-2.77)`

Yields the absolute value (value without regard to sign) of the argument. The example yields `2.77`. [2.4.1]

**ASC**

*Syntax:* `ASC(sexpr)`  
*Example:* `ASC("QUEST II")`

Yields the ASCII code for the first character in the argument. The example yields `81` (ASCII code for Q). [4.2.5, C]

**Assignment Statement**

*Syntax:* `[LET] avar = aexpr  
[LET] svar = sexpr`  
*Example:* `LET A = 23.567  
A$ = "HUMBUG"`

Assigns the value of the expression following `=` to the variable preceding it. LET is optional. [2.2]
ATN
Syntax: \texttt{ATN(aexpr)}
Example: \texttt{ATN(\,8.771\,)}

Yields the arc tangent, in radians, of the argument. The example yields \texttt{720001187} (radians). \[2.4.1\]

CALL
Syntax: \texttt{CALL aexpr}
Example: \texttt{CALL -922}

Executes a machine-language subroutine at the specified decimal memory address. The example issues a line feed. \[7.1.3, F\]

CHR$
Syntax: \texttt{CHR\$(aexpr)}
Example: \texttt{CHR\$(65)}

Yields the character corresponding to the ASCII code given as an argument. The example yields the letter A. \[4.2.5\]

CLEAR
Syntax: \texttt{CLEAR}
Example: \texttt{CLEAR}

Resets all variables and internal control information to their initial state. Program code is unaffected. \[1.2.2\]

COLOR =
Syntax: \texttt{COLOR = aexpr}
Example: \texttt{COLOR = 12}

Sets the display color for plotting low-resolution graphics. The example sets the display color to green. \[6.1.2\]

CONT
Syntax: \texttt{CONT}
Example: \texttt{CONT}

Resumes program execution after it has been halted by \texttt{STOP}, \texttt{END}, \texttt{\[CONTROL\]-C}, or (sometimes) \texttt{\[CONTROL\]-RESET}. \[1.3.3\]
**COS**

*Syntax:*  \( \text{COS} \ (\text{aexpr}) \)

*Example:*  \( \text{COS} \ (\text{2}) \)

Yields the cosine of the argument, which must be expressed in radians. The example yields \(-0.416146836\). [2.4.1]

**DATA**

*Syntax:*  \( \text{DATA} \ [\text{literal} \ | \ \text{string} \ | \ \text{real} \ | \ \text{integer}] \)

*Example:*  \( \text{DATA} \ \text{JOHN SMITH, "CODE 32", 23, 45, -6} \)

Creates a list of items for use by READ statements. In the example, the first item is the string "JOHN SMITH", the second is the string "CODE 32", the third is the real number 23.45, and the fourth is the integer -6. [5.1.4]

**DEF FN**

*Syntax:*  \( \text{DEF FN name} \ (\text{name}) = \text{aexpr} \)

*Example:*  \( \text{DEF FN CUBE} \ (\text{X}) = \text{X * X * X} \)

Defines a new function for use in the program. The example defines a function that yields the cube of its argument. [2.4.3]

**DEL**

*Syntax:*  \( \text{DEL} \ \text{linenum}, \text{linenum} \)

*Example:*  \( \text{DEL} \ 23, 56 \)

Deletes a range of consecutive lines from the program. The example deletes all lines numbered between 23 and 56, inclusive. [1.1.5]

**DIM**

*Syntax:*  \( \text{DIM} \ \text{name}[\%|\$] \ \text{subscript}[\{, \text{name}[\%|\$] \ \text{subscript}\}] \)

*Example:*  \( \text{DIM} \ \text{MARK} \ (50, 3), \ \text{NAME}\$ \ (50) \)

Defines and allocates space for one or more arrays. The example defines a two-dimensional real array MARK, whose first subscript varies from 0 to 50 and whose second varies from 0 to 3, and a string array NAME\$ with one subscript that varies from 0 to 50. [4.1.1]

Summary of Applesoft Statements and Functions
**DRAW**

Syntax: `DRAW aexpr [AT aexpr , aexpr]

Example: `DRAW 4 AT 50 , 100
DRAW 4`

Draws a shape at a specified point on the high-resolution graphics screen from the shape table currently in memory. The first example draws shape number 4, beginning in column 50, row 100, using the current color, scale, and rotation settings; the second example draws shape 4 at the last point plotted by `HPLOT`, `DRAW`, or `XDRAW`. [6.3.2]

**END**

Syntax: `END`

Example: `END`

Terminates the execution of the program and returns control to the user. No message is displayed. [3.6.2]

**EXP**

Syntax: `EXP (aexpr)`

Example: `EXP (2)`

Yields the mathematical exponential of its argument (the constant $e = 2.7182818$ raised to the power specified by the argument). The example yields $e$ squared, or $7.3890561$. [2.4.1]

**FLASH**

Syntax: `FLASH`

Example: `FLASH`

Causes all text displayed on the screen with subsequent `PRINT` statements to flash between white-on-black and black-on-white. May not work properly for lowercase letters (and other characters with ASCII codes above 95) if the 80-Column Text Card is installed and running in "active-80" mode. [5.2.4]
FN
Syntax: FN name (aexpr)
Example: FN CUBE (6)
Applies a designated function to the value of the argument expression. Assuming the definition for the function CUBE given above under DEF FN, the example yields the value 216. [2.4.3]

FOR
Syntax: FOR name = aexpr TO aexpr [STEP aexpr]
Example: FOR J = 1 TO 10
FOR MARK = 0 TO 100 STEP 5
FOR NUMBER = 20 TO -20 STEP -2
Marks the beginning of a loop, identifies the index variable, and gives the variable's starting and ending values and (optionally) the amount by which it is to change (step) on each pass through the loop. The first example begins a loop whose index variable J will take on all values from 1 to 10, stepping by 1; the second begins a loop whose index variable MARK will take on values from 0 to 100, stepping by 5; the third begins a loop whose index variable NUMBER will take on values from 20 to -20, stepping by -2. [3.3.1]

FRE
Syntax: FRE (expr)
Example: FRE (0)
Yields the amount of remaining memory, in bytes, available to the program. The argument is ignored, but must be a valid Applesoft expression. [7.2.3]

GET
Syntax: GET var
Example: GET ANSWER$
Accepts a single character from the keyboard without displaying it on the screen and without requiring that the RETURN key be pressed. Program execution is suspended until the user presses a key. In the example, the character typed will be assigned to the variable ANSWER$. [5.1.3]
GOSUB
Syntax:  GOSUB linenum
Example: GOSUB 250
Executes a subroutine beginning at the designated line number
(250 in the example). [3.4.1]

GOTO
Syntax:  GOTO linenum
Example: GOTO 400
Sends control unconditionally to the designated line number (400 in
the example). [3.1]

GR
Syntax:  GR
Example: GR
Converts the display to 40 rows of low-resolution graphics with four
lines of text at the bottom. The screen is cleared to black, the cursor
is moved to the beginning of the last line, and the low-resolution
display color is set to black. [6.1.1]

HCOLOR =
Syntax:  HCOLOR = aexpr
Example: HCOLOR = 1
Sets the display color for plotting high-resolution graphics. The
example sets the display color to green. [6.2.3]

HGR
Syntax:  HGR
Example: HGR
Converts the display to 160 rows of high-resolution graphics with four
lines for text at the bottom. The screen is cleared to black and page 1
of high-resolution graphics is displayed. The contents of the text dis-
play, the location of the cursor, and the high-resolution display color
are unaffected. [6.2.1]
HGR2

Syntax: HGR2
Example: HGR2

Converts the display to full-screen (192 rows) high-resolution graphics with no text. The screen is cleared to black and page 2 of high-resolution graphics is displayed. The contents of the text display, the location of the cursor, and the high-resolution display color are unaffected. [6.2.2]

HIMEM:

Syntax: HIMEM: aexpr
Example: HIMEM: 32767

Sets the address of the highest memory location available to the Applesoft program, including its variables. The example sets the end of program and variable storage to 32767. Used to protect an area of memory for data, high-resolution graphics, or machine-language code. [7.2.1]

HLIN

Syntax: HLIN aexpr1, aexpr2 AT aexpr3
Example: HLIN 10, 20 AT 30

Draws a horizontal line in low-resolution graphics, using the current low-resolution display color. The example draws a line across row 30 from column 10 to column 20. [6.1.4]

HOME

Syntax: HOME
Example: HOME

Clears all text from the text window and moves the cursor to the top-left corner of the window. [5.2.4]
H PLOT

Syntax:  

```
H PLOT aexpr , aexpr [{TO aexpr , aexpr}]
H PLOT TO aexpr , aexpr [{TO aexpr , aexpr}]
```

Example:  

```
H PLOT 75, 20
H PLOT 48, 115 TO 79, 84 TO 110, 115
H PLOT TO 270, 10
```

Plots a point or line on the high-resolution graphics screen in the current high-resolution display color. The first example plots a single point at column 75, row 20; The second example draws lines from column 48, row 115 to column 79, row 84 to column 110, row 115; the third draws a line to column 270, row 10 from the last point plotted with H PLOT, using the color of the last point plotted (not necessarily the current display color). [6.2.4]

H T A B

Syntax:  

```
H T A B aexpr
```

Example:  

```
HTAB 23
```

Positions the cursor to a specified column of the text display. The example moves the cursor to column 23. If you have the Apple IIe 80-Column Text Card, see the manual accompanying that product for further information on using HTAB. [5.2.4]

I F ... T H E N

Syntax:  

```
I F expr THEN statement [{: statement}]
I F expr THEN [G O T O] linenum
I F expr [THEN] G O T O linenum
```

Example:  

```
IF AGE < 18 THEN A = 0 : B = 1 :
   C = 2
IF ANSWER$ = "YES" THEN GOTO 100
IF N > MAX THEN GOTO 25
IF N > MAX THEN 25
IF N > MAX GOTO 25
```

Executes or skips one or more statements, depending on the truth of a stated condition. The first example sets A to 0, B to 1, and C to 2 if the value of AGE is less than 18; the second branches to line 100 if the value of ANSWER$ is the string "YES"; the last three all branch to line 25 if the value of N is greater than that of MAX. In all cases, if the stated condition is false, execution continues with the next program line. [3.2.2]
IN#

Syntax: IN# aexpr
Example: IN# 2

Specifies the source for subsequent input. The example causes subsequent input to be read from the device in expansion slot 2. [5.1.1]

INPUT

Syntax: INPUT [sexpr ;] var [{, var}]
Example: INPUT A% INPUT "TYPE AGE, THEN A COMMA, THEN NAME" ; AGE, NAME$

Reads a line of input from the current input device. The first example reads a value into variable A%; the second displays a prompting message and then reads values into variables AGE and NAME$. [5.1.2]

INT

Syntax: INT (aexpr)
Example: INT (98.6) INT (-273.16)

Yields the integer part of the argument value. The examples yield 98 and -274, respectively. [2.4.1]

INVERSE

Syntax: INVERSE
Example: INVERSE

Causes all text displayed on the screen with subsequent PRINT statements to appear in black-on-white instead of the usual white-on-black. May not work properly for lowercase letters (and other characters with ASCII codes above 95) if the 80-Column Text Card is installed and running in "active-80" mode. [5.2.4]

LEFT$

Syntax: LEFT$ (sexpr, aexpr)
Example: LEFT$ ("APPLESOFT", 5)

Yields a specified number of characters from the beginning of a string. The example yields the string "APPLE". [4.2.4]
LEN
Syntax: LEN (sexpr)
Example: LEN ("NEVER A DULL MOMENT")
Yields the length of a string in characters. The example yields 19. [4.2.2]

LET
See “Assignment Statement,” above.

LIST
Syntax: LIST [linenum1][-linenum2]
LIST [linenum1],[linenum2]
Example: LIST
   LIST 150
   LIST 200-300
   LIST 200,300
Displays all or part of the program on the screen, or writes it to the current output device. The first example lists the entire program; the second lists line 150 only; the last two list all lines numbered from 200 to 300, inclusive. [1.2.3]

LOAD
Syntax: LOAD [name]
Example: LOAD
   LOAD DEMO
Reads a program into memory from a disk or tape. The first example reads a program from a tape cassette; the second reads from a disk file named DEMO. If you have one or more disk drives, see your DOS manual for further information. [1.2.6, M]

LOG
Syntax: LOG (aexpr)
Example: LOG (2)
Yields the natural logarithm of the argument. The example yields 0.693147181. [2.4.1]
LOMEM:
Syntax:  LOMEM: aexpr
Example: LOMEM: 24576
Sets the address of the lowest memory location available to the program for variable storage. The example sets the beginning of variable storage to 24576. [7.2.2]

MID$
Syntax:  MID$(sexpr, aexpr [, aexpr])
Example: MID$("AN APPLE A DAY", 4, 5)
MID$("AN APPLE A DAY", 4)
Yields a specified number of characters beginning at a specified position in a given string. The first example yields the string "APPLE"; the second yields the string "APPLE A DAY". [4.2.4]

NEW
Syntax:  NEW
Example:  NEW
Clears the current program from memory and resets all variables and internal control information to their initial states. [1.2.1]

NEXT
Syntax:  NEXT [avar {, avar}]
Example:  NEXT
    NEXT INDEX
    NEXT J, I
Marks the end of a loop and causes the loop to be repeated for the next value of the index variable, as specified in the corresponding FOR statement. The first example ends the most recently entered loop; the second ends the loop whose index variable is INDEX; the third ends the pair of nested loops whose index variables are J and I. [3.3.2]
NORMAL

Syntax: NORMAL
Example: NORMAL

Causes all text displayed on the screen with subsequent PRINT statements to appear in the usual white-on-black; cancels the effects of INVERSE or FLASH. [5.2.4]

NOTRACE

Syntax: NOTRACE
Example: NOTRACE

Stops the display of line numbers for each statement executed; cancels the effects of TRACE. [7.3.2]

ON...GOSUB

Syntax: ON aexpr GOSUB linenum [{, linenum}]
Example: ON ID GOSUB 100, 200, 23, 4005, 500

Chooses a subroutine to execute depending on the value of an expression. The example transfers control to the subroutine beginning at line 100, 200, 23, 4005, or 500, depending on whether the value of ID is 1, 2, 3, 4, or 5; if ID has none of these values, execution continues with the next statement. [3.4.3]

ON...GOTO

Syntax: ON aexpr GOTO linenum [{, linenum}]
Example: ON ID GOTO 100, 200, 23, 4005, 500

Chooses a line number to branch to depending on the value of an expression. The example transfers control to line 100, 200, 23, 4005, or 500, depending on whether the value of ID is 1, 2, 3, 4, or 5; if ID has none of these values, execution continues with the next statement. [3.2.1]

ONERR GOTO

Syntax: ONERR GOTO linenum
Example: ONERR GOTO 500

Replaces Applesoft's normal error-handling mechanism with a subroutine beginning at a specified line number. The example establishes an error-handling subroutine beginning at line 500. [3.5.1, E]
PDL
Syntax: PDL (aexpr)
Example: PDL (1)
Reads the current dial setting on a designated hand control. The example reads the dial on hand control 1. [5.1.6, F.4]

PEEK
Syntax: PEEK (aexpr)
Example: PEEK (37)
Yields the contents of a specified location in memory. The example yields the contents of location 37, which contains the current vertical position of the text cursor on the display screen. [7.1.1, F.1]

PLOT
Syntax: PLOT aexpr, aexpr
Example: PLOT 10, 20
Plots a single block of the current display color at a specified position on the low-resolution graphics screen. The example plots a block at column 10, row 20. [6.1.3]

POKE
Syntax: POKE aexpr, aexpr
Example: POKE -16302, 0
Stores a value into a specified location in memory. The example stores the value 0 at location 49234 (65536 - 16302), causing the display to switch from mixed graphics and text to full-screen graphics. [7.1.2, F]

POP
Syntax: POP
Example: POP
Removes the most recent return address from the control stack, causing the next RETURN statement to send control to the statement following the second most recently executed GOSUB. [3.4.4]
POS
Syntax: POS (expr)
Example: POS (0)

Yields the current horizontal position of the cursor on the text display. The argument is ignored, but must be a valid Applesoft expression. [5.2.4]

PR#
Syntax: PR# aexpr
Example: PR# 1

Specifies the destination for subsequent output. The example causes subsequent output to be sent to the device in expansion slot 1. [5.2.1]

PRINT
Syntax: PRINT [{expr [; ; i]}]
Example: PRINT
PRINT A$, "X = " ; X

 Writes a line of output to the current output device. The first example writes a blank line; the second writes the value of variable A $, followed at the next available tab position by the string " X = " , followed immediately by the value of variable X. [5.2.2]

READ
Syntax: READ var [{, var}]
Example: READ A, B%, C$

Reads values from DATA statements in the body of the program. The example reads values into variables A, B %, and C $. [5.1.4]

RECALL
Syntax: RECALL name [ ; ]
Example: RECALL MX

Reads values into an array from a tape cassette. The example reads values into array MX. [M]
REM
Syntax:  REM {character}
Example: REM THIS A REMARK

Includes remarks in the body of a program for the benefit of a human reader. [1.1.7]

RESTORE
Syntax:  RESTORE
Example: RESTORE

Causes the next READ statement executed to begin reading at the first item of the first DATA statement in the program. [5.1.5]

RESUME
Syntax:  RESUME
Example: RESUME

At the end of an error-handling routine (see ONERR GOTO), causes resumption of the program at the beginning of the statement in which the error occurred. [3.5.2]

RETURN
Syntax:  RETURN
Example: RETURN

Returns control from a subroutine to the statement following the GOSUB that called the subroutine. [3.4.2]

RIGHT$
Syntax:  RIGHT$ (sexpr, aexpr)
Example: RIGHT$ ("APPLESOFT", 4)

Yields a specified number of characters from the end of a string. The example yields the string "SOFT". [4.2.4]

RND
Syntax:  RND (aexpr)
Example: RND (1)

Yields a random number between 0 and 1. Zero and negative argument values yield repeatable sequences of random numbers. [2.4.2]
ROT =

Syntax:  ROT = aexpr
Example:  ROT = 16

Sets the angular rotation for high-resolution shapes to be drawn with Draw or XDraw. The example causes the shape to be rotated 90 degrees clockwise. [6.3.2]

RUN

Syntax:  RUN [linenum | name]
Example:  RUN
RUN 500
RUN DEMO

Executes an Applesoft program. The first example executes the program currently in memory from the beginning; the second executes the program in memory, starting at line 500; the third loads and executes a program from a disk file named DEMO. [1.2.4]

SAVE

Syntax:  SAVE [name]
Example:  SAVE
SAVE DEMO

Writes the Applesoft program currently in memory to a disk or tape. The first example writes the program to a tape cassette; the second writes it to a disk file named DEMO. [1.2.5, M]

SCALE=

Syntax:  SCALE = aexpr
Example:  SCALE = 10

Sets the scale factor for high-resolution shapes to be drawn with Draw or XDraw. The example causes the shape to be drawn ten times bigger than the definition given in the shape table. [6.3.2]
SCRN
Syntax: `SCRN (aexpr, aexpr)`
Example: `SCRN (10, 20)`
Yields the code for the color currently displayed at a designated position on the low-resolution graphics screen. The example yields the code for the color at column 10, row 20. [6.1.6]

SGN
Syntax: `SGN (aexpr)`
Example: `SGN (-144)`
Yields a value of -1, 0, or +1, depending on the sign of the argument. The example yields -1. [2.4.1]

SHLOAD
Syntax: `SHLOAD`
Example: `SHLOAD`
Loads a shape table into memory from a tape cassette. [6.3.2, M]

SIN
Syntax: `SIN (aexpr)`
Example: `SIN (2)`
Yields the sine of the argument, which must be expressed in radians. The example yields 0.909297427. [2.4.1]

SPC
Syntax: `SPC (aexpr)`
Example: `SPC (8)`
Introduces a specified number of spaces into the line being written by a PRINT statement. The example writes 8 spaces. [5.2.4]

SPEED=
Syntax: `SPEED = aexpr`
Example: `SPEED = 50`
Sets the rate at which text characters are sent to the display screen or other input/output device. The slowest rate is 0; the fastest is 255. [5.2.4]

Summary of Applesoft Statements and Functions
SQR
Syntax: `SQR (aexpr)`
Example: `SQR (2)`

Yields the positive square root of the argument; the example yields 1.41421356. [2.4.1]

STOP
Syntax: `STOP`
Example: `STOP`

Terminates the execution of the program and returns control to the user. A message is displayed identifying the program line in which the STOP statement appears. [3.6.1]

STORE
Syntax: `STORE name[%,]`
Example: `STORE MX`

Stores values from an array onto a tape cassette. The example stores the contents of array MX. [M]

STR$
Syntax: `STR$ (aexpr)`
Example: `STR$ (12.45)`

Yields a string representing the numeric value of the argument. The example yields the string "12.45". [4.2.5]

TAB
Syntax: `TAB (aexpr)`
Example: `TAB (23)`

Positions the text cursor to a specified position on the output line during execution of a PRINT statement. The example moves the cursor to column 23. [5.2.4]
TAN
Syntax: TAN (aexpr)
Example: TAN (2)
Yields the tangent of the argument, which must be expressed in radians. The example yields \(-2 \times 18503987.241\).

TEXT
Syntax: TEXT
Example: TEXT
Converts the display to 24 lines of text, with the cursor positioned at the beginning of the bottom line. [5.2.4]

TRACE
Syntax: TRACE
Example: TRACE
Causes the line number of each statement to be displayed on the screen as it is executed. [7.3.1]

USR
Syntax: USR (aexpr)
Example: USR (3)
Executes a machine-language subroutine supplied by the user, passing it a specified argument. The subroutine is entered via a JMP (Jump) instruction stored at addresses \(0A\) through \(0C\) hexadecimal. The example passes the argument value 3. [7.1.4]

VAL
Syntax: VAL (sexpr)
Example: VAL ("-3.7E4")
Yields the numeric value represented by the string supplied as an argument. The example yields \(-37000\). [4.2.5]
**VLIN**

Syntax: `VLIN aexpr , aexpr AT aexpr`

Example: `VLIN 10, 20 AT 30`

Draws a vertical line in low-resolution graphics, using the current low-resolution display color. The example draws a line down column 30 from row 10 to row 20. [6.1.5]

**VTAB**

Syntax: `VTAB aexpr`

Example: `VTAB 15`

Positions the cursor to a specified row of the text display. The example moves the cursor to row 15. [5.2.4]

**WAIT**

Syntax: `WAIT aexpr , aexpr [, aexpr]`

Example: `WAIT 49347, 15
WAIT 49347, 15, 12`

Suspends program execution until a specified bit pattern appears at a specified memory location. Used to wait for a status signal from a peripheral device. The second and (optional) third arguments are masks: the second specifies which bits of the designated location are of interest, the third specifies the values to be tested for in those bits. The first example suspends execution until a one bit appears in any of the four low-order bit positions of location 49347; the second waits for a one bit in position 0 or 1 or a zero bit in position 2 or 3. [7.1.5]

**XDRAW**

Syntax: `XDRAW aexpr [ AT aexpr , aexpr]`

Example: `XDRAW 4 AT 50, 100
XDRAW 4`

Draws a shape from the shape table currently in memory at a specified point on the high-resolution graphics screen. Each point in the shape is plotted using the complement of the color currently displayed at that point. Typically used to erase a shape already drawn. The first example erases shape number 4, beginning in column 50, row 100, using the current scale and rotation settings; the second example erases shape 4 at the last point plotted by HPLOT, DRAW, or XDRAW. [6.3.2]
Terms used in the syntax definitions in Appendix A are defined below. The following symbols are used in the syntax definitions:

- `:=` means "is defined as"
- `|` separates alternative definitions
- (alternative definitions for the same term may also be given separately)
- `[]` enclose elements that may be omitted
- `{}` enclose elements that may be repeated one or more times

**aexpr (arithmetic expression)**

- `:=` real | integer | avar | fcall
- `:=` unop aexpr
- `:=` aexpr alop aexpr
- `:=` sexpr relop sexpr
- `:=` ( aexpr )

Parentheses may not be nested more than 36 levels deep.

**alop (arithmetic or logical operator)**

- `:=` aop | relop | lop

**aop (arithmetic operator)**

- `:=` + | - | * | / | ^

**avar (arithmetic variable)**

- `:=` realvar | intvar

**character**

- `:=` letter | digit | spchar | quote | space

**digit**

- `:=` 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
expr (expression)
    ::= aexpr | sexpr

fcall (function call)
    ::= name ( expr [ { , expr } ] )

integer
    ::= [ + | - ]{digit}

Valid integers must be between -32767 and +32767.

intvar (integer variable)
    ::= name% [subscript]

letter
    ::= uppercase | lowercase

line
    ::= linenum [ {statement ; } ] statement RETURN

linenum (line number)
    ::= digit [ {digit} ]

Line numbers must be in the range 0 to 63999.

literal
    ::= [ {character} ]

lop (logical operator)
    ::= AND | OR

Notice that NOT is not included here.

lowercase
    ::= a | b | c | d | e | f | g | h | i | j | k |
    | l | m | n | o | p | q | r | s | t | u | v |
    | w | x | y | z

name
    ::= uppercase [ {uppercase | digit} ]

A name may be of any length. When distinguishing one
name from another, Applesoft ignores any characters after
the first two. However, even the ignored portion of a name
must not contain a special character or any of Applesoft's
reserved words.
quote
::= "

real
::= [+ | -] {digit} [ . {digit} ] [E [+ | -] {digit} {digit}]
::= [+ | -] [{digit}] [ . {digit} ] [E [+ | -] {digit} {digit}]

The letter E in a real number stands for "times ten to the power." Valid reals must be between -1E38 and +1E38.

Applesoft recognizes the following as reals and evaluates them as zero:

In addition, the following are recognized as reals and evaluated as zero when used as numeric responses to INPUT or as numeric elements of DATA:
SPACE + - E + E - E E+
E- + E+ + E- - E+ - E-

The GET statement evaluates all of the single-character reals listed above as zero.

realvar (real variable)
::= name [subscript]

relop (relational operator)
::= = | < | > | <= | => | >= | => | >> | <<

schar (string character)
::= letter | digit | spchar | space

Notice that the quote character (" ) is not included here.

sexpr (string expression)
::= string | svar | sfcall
::= sexpr sop sexpr
::= ( sexpr )

Parentheses may not be nested more than 36 levels deep.
sfcall (string function call)
    := name$ ( expr [{, expr}] )

sop (string operator)
    := +

space
    := SPACE

spchar (special character)
    := + | - | * | / | ^ | = | < | > | ( | ) |
         | , | . | : | ; | % | $ | # | ? | & | '
         | @ | ! | [ | ] | { | } | \ | | | | |

Control characters (characters typed while holding down the CONTROL key) and the null character are also considered special characters. Notice that the quote character (") and SPACE are not included here.

statement
    See Appendix A for syntactic definitions of all Applesoft statements.

string
    := "[[schar]]"
    := "[[schar]]"

The second form of string can appear only at the end of a line.

subscript
    := ( aexpr [{, aexpr} ] )

The maximum number of dimensions (aexpr's) is 89, although in practice this is limited by the extent of memory available.

svar (string variable)
    := name$ [subscript]

unop (unary operator)
    := + | - | NOT
uppercase
    ::= A | B | C | D | E | F | G | H | I | J | K |
    L | M | N | O | P | Q | R | S | T | U | V |
    W | X | Y | Z

var (variable)
    ::= avar | svar
Below is a chart of the ASCII (American Standard Code for Information Interchange) character codes. The first 32 codes represent control characters; to type these characters from the Apple IIe keyboard, press the CONTROL key and hold it down while pressing the designated character. (Some of these characters also have single-key representations, as noted.) The abbreviations given for these codes in the column labeled “Char” represent standard control functions originally intended for use on teletypes; the meanings of these abbreviations are given in the “Meaning” column. Functions marked with an asterisk (*) are implemented on the Apple IIe; the others are listed purely for historical interest.

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>NUL</td>
<td>CONTROL</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>SOH</td>
<td>CONTROL</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>STX</td>
<td>CONTROL</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>ETX</td>
<td>CONTROL</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>EOT</td>
<td>CONTROL</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>ENQ</td>
<td>CONTROL</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>ACK</td>
<td>CONTROL</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>BEL</td>
<td>CONTROL</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>BS</td>
<td>CONTROL</td>
<td>H or LEFT-ARROW</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>HT</td>
<td>CONTROL</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>LF</td>
<td>CONTROL</td>
<td>J or DOWN-ARROW</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>VT</td>
<td>CONTROL</td>
<td>K or UP-ARROW</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>FF</td>
<td>CONTROL</td>
<td>L</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Char</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>---------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>CR</td>
<td>CONTROL -M</td>
<td>* carriage return</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>SD</td>
<td>CONTROL -N</td>
<td>shift out</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>SI</td>
<td>CONTROL -D</td>
<td>shift in</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>DLE</td>
<td>CONTROL -P</td>
<td>data link escape</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>DC1</td>
<td>CONTROL -Q</td>
<td>device link escape</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>DC2</td>
<td>CONTROL -R</td>
<td>device control 1</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>DC3</td>
<td>CONTROL -S</td>
<td>device control 2</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>DC4</td>
<td>CONTROL -T</td>
<td>device control 3</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>NAK</td>
<td>CONTROL -U</td>
<td>shift out</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>SYN</td>
<td>CONTROL -V</td>
<td>device control 4</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>ETB</td>
<td>CONTROL -W</td>
<td>end of transmission block</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>CAN</td>
<td>CONTROL -X</td>
<td>* cancel</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>EM</td>
<td>CONTROL -Y</td>
<td>end of medium</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>SUB</td>
<td>CONTROL -Z</td>
<td>substitute</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>ESC</td>
<td>CONTROL -[</td>
<td>escape</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>FS</td>
<td>CONTROL -\</td>
<td>file separator</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>GS</td>
<td>CONTROL -]</td>
<td>group separator</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>RS</td>
<td>CONTROL -^</td>
<td>record separator</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>US</td>
<td>CONTROL -_</td>
<td>unit separator</td>
</tr>
</tbody>
</table>

The following characters can be typed directly from the keyboard:

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>20</td>
<td>SPACE</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
<td>,</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**ASCII Character Codes**
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>58</td>
<td>3A</td>
<td>:</td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td>=</td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>C</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>D</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>F</td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>G</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>H</td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>74</td>
<td>4A</td>
<td>J</td>
</tr>
<tr>
<td>75</td>
<td>4B</td>
<td>K</td>
</tr>
<tr>
<td>76</td>
<td>4C</td>
<td>L</td>
</tr>
<tr>
<td>77</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>78</td>
<td>4E</td>
<td>N</td>
</tr>
<tr>
<td>79</td>
<td>4F</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td>P</td>
</tr>
<tr>
<td>81</td>
<td>51</td>
<td>Q</td>
</tr>
<tr>
<td>82</td>
<td>52</td>
<td>R</td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td>S</td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td>T</td>
</tr>
<tr>
<td>85</td>
<td>55</td>
<td>U</td>
</tr>
<tr>
<td>86</td>
<td>56</td>
<td>V</td>
</tr>
<tr>
<td>87</td>
<td>57</td>
<td>W</td>
</tr>
<tr>
<td>88</td>
<td>58</td>
<td>X</td>
</tr>
<tr>
<td>89</td>
<td>59</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>5A</td>
<td>Z</td>
</tr>
<tr>
<td>91</td>
<td>5B</td>
<td>[</td>
</tr>
<tr>
<td>92</td>
<td>5C</td>
<td>\</td>
</tr>
<tr>
<td>93</td>
<td>5D</td>
<td>]</td>
</tr>
<tr>
<td>94</td>
<td>5E</td>
<td>^</td>
</tr>
<tr>
<td>Dec</td>
<td>Hex</td>
<td>Char</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>95</td>
<td>5F</td>
<td>_</td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td>'</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>127</td>
<td>7F</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
Reserved Words

Following is a list of Applesoft’s reserved words. In most cases, these character sequences cannot be used as, or embedded in, variable names (but see the comments at the end of the list).

<table>
<thead>
<tr>
<th>ABS</th>
<th>AND</th>
<th>ASC</th>
<th>AT</th>
<th>ATN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>CHR$</td>
<td>CLEAR</td>
<td>COLOR=</td>
<td>CONT</td>
</tr>
<tr>
<td>DATA</td>
<td>DEF</td>
<td>DEL</td>
<td>DIM</td>
<td>DRAW</td>
</tr>
<tr>
<td>END</td>
<td>EXP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH</td>
<td>FN</td>
<td>FOR</td>
<td>FRE</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>GOSUB</td>
<td>GOTO</td>
<td>GR</td>
<td></td>
</tr>
<tr>
<td>HCOLOR=</td>
<td>HGR</td>
<td>HGR2</td>
<td>HIMEM:</td>
<td>HLIN</td>
</tr>
<tr>
<td>LEFT$</td>
<td>LEN</td>
<td>LET</td>
<td>LIST</td>
<td>LOAD</td>
</tr>
<tr>
<td>MID$</td>
<td>NEXT</td>
<td>NORMAL</td>
<td>NOT</td>
<td>NOTRACE</td>
</tr>
<tr>
<td>NEW</td>
<td>ONERR</td>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>PDL</td>
<td>PEEK</td>
<td>PRINT</td>
<td>PLOT</td>
</tr>
<tr>
<td>PRINT</td>
<td>POKE</td>
<td>POP</td>
<td>POS</td>
<td>PR#</td>
</tr>
<tr>
<td>READ</td>
<td>RECALL</td>
<td>REM</td>
<td>RESTORE</td>
<td>RESUME</td>
</tr>
<tr>
<td>RND</td>
<td>ROT=</td>
<td>RUN</td>
<td>RETURN</td>
<td>RIGHT$</td>
</tr>
<tr>
<td>SAVE</td>
<td>SCALE=</td>
<td>SCRN</td>
<td>SGN</td>
<td>SHLOAD</td>
</tr>
<tr>
<td>SPEED=</td>
<td>SQR</td>
<td>STEP</td>
<td>STOP</td>
<td>SIN</td>
</tr>
<tr>
<td>TAB</td>
<td>TAN</td>
<td>TEXT</td>
<td>THEN</td>
<td>TO</td>
</tr>
<tr>
<td>USR</td>
<td>VLIN</td>
<td>XDRAW</td>
<td>XPLOT</td>
<td></td>
</tr>
<tr>
<td>VAL</td>
<td>VTAB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td>VLIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XDRAW</td>
<td>XPLOT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Applesoft *tokenizes* these reserved words (converts them into one-byte internal codes); see Section H.4 for a list of tokens. All other characters in a program occupy one byte each of program storage.

The ampersand character ( &) is reserved for Applesoft's internal use and for user-supplied machine-language routines. When executed as an instruction, it causes a JSR to address $03F5 hexadecimal.

*X PLOT* is a reserved word that does not correspond to a current Applesoft statement.

Some reserved words are recognized by Applesoft only in certain contexts:

- **COLOR, HCOLOR, ROT, SCALE, and SPEED** are interpreted as reserved words only if the next non-space character is an equal sign (=). This is of little benefit in the case of COLOR and HCOLOR, as the embedded reserved word OR prevents their use as variable names anyway.
- **HIMEM and LOMEM** are interpreted as reserved words only if the next non-space character is a colon (:).
- **IN and PR** are interpreted as reserved words only if the next non-space character is a number sign (#).
- **SCRN, SPC, and TAB** are interpreted as reserved words only if the next non-space character is a left parenthesis, (.
- **ATN** is interpreted as a reserved word only if there is no space between the T and the N. If a space occurs between the T and the N, the reserved word AT is interpreted instead of ATN.
- **TO** is interpreted as a reserved word unless preceded by an A and there is a space between the T and the O. In that case, the reserved word AT is interpreted instead of TO.

Even if you don't embed reserved words in your variable names, they can sometimes pop up unexpectedly and cause problems. For example, the statement

```
100 FOR A = LOFT OR LEFT TO 15
```

is interpreted as

```
100 FOR A = LOF TO RLEFT TO 15
```

and causes a syntax error. To force the correct interpretation, use parentheses:

```
100 FOR A = (LOFT) OR (LEFT) TO 15
```
Below is a list of Applesoft’s error messages and their causes. When an error occurs in immediate execution, Applesoft sounds a “beep” and displays a message of the form

`?XX ERROR`

where `XX` is the name of the particular error, as listed below.

In deferred execution (during the course of a running program), the message takes the form

`?XX ERROR IN YY`

where `YY` is the line number of the statement in which the error occurred; the Applesoft prompt character (J) and the cursor are displayed, and control of the system is returned to the user. Variable values and the text of the program remain intact, but internal control information is erased and the program cannot be continued with the CONT command (see Section 1.3.3). An error in a deferred-execution statement is not detected until the statement is executed.

The error handling described above can be overridden by an error-handling routine in the program itself, established with the ONERR GOTO statement (see Section 3.5.1). Error codes for use in such an error-handling routine are given below in square brackets following the error names. When an error occurs, the code listed is stored at location 222 decimal; it can be retrieved from that location with the PEEK function (see Section 7.1.1). Errors for which no code is given cannot occur in deferred execution.

Errors associated with the Disk Operating System (DOS) will also register at location 222; see the DOS manual for further information.
Debugging suggestions given below under the individual error messages are not intended to be exhaustive or comprehensive; the causes of program bugs are numberless as the sands of the sea and the stars of the sky.

**BAD SUBSCRIPT [107]**
A reference was made to an array element that is outside the dimensions of the array. This error can occur if the wrong number of dimensions is used in an array reference: for instance,

\[
\text{LET } A(1, 1, 1) = Z
\]

when \(A\) has been defined by

\[
\text{DIM } A(2, 2)
\]

**CAN'T CONTINUE**
An attempt was made to continue a program with the CONT command when no program exists in memory, or after an error, or after a line has been changed, deleted from, or added to the program.

**DIVISION BY ZERO [133]**
An attempt was made to divide by zero; division by zero is mathematically undefined. Often occurs when a variable is used in an arithmetic expression before begin given a value (all numeric variables initially have the value zero). To debug, examine the divisor of the expression where the error occurred to see why it unexpectedly has a zero value. Look particularly for variables that have inadvertently been used without having been given a nonzero value.

**FORMULA TOO COMPLEX [191]**
More than two statements of the form

\[
\text{IF } "ZZ" \text{ THEN }
\]

were executed (where "ZZ" is any quoted string). The Applesoft IF...THEN statement wasn't intended to be used with strings, and the results of such statements are not meaningful. The wisest policy is to avoid this type of construction altogether.
ILLEGAL DIRECT

An attempt was made to use one of the following statements in immediate execution:

- DEF FN
- GET
- INPUT'
- ONERR GOTO
- READ
- RESUME

ILLEGAL QUANTITY [53]

The argument supplied to a statement or function was out of the allowed range. This error can be caused by

- a negative array subscript (for example, LET A (-1) = 0)
- LOG with a negative or zero argument
- SQR with a negative argument
- A ^ B with A negative and B not an integer
- use of LEFT$, MID$, RIGHT$, WAIT, PEEK, POKE, CALL, TAB, SPC, ON...GOTO, ON...GOSUB, or any of the graphics statements or functions with an improper argument

NEXT WITHOUT FOR [0]

The variable named in a NEXT statement did not agree with the variable in the corresponding FOR statement, or a nameless NEXT was executed when no FOR was in effect. The most common causes of this error are forgetting a FOR or NEXT statement, typing the wrong variable in the NEXT statement, crossing loops, or accidentally branching into the body of a FOR loop.
OUT OF DATA [42]
A READ statement was executed after all DATA statements in the program had already been read. A READ statement may have been executed more times than intended (for example, in an infinite loop), or one or more DATA statements may have been inadvertently omitted. Sometimes caused by accidentally leaving out a RESTORE statement.

OUT OF MEMORY [77]
Any of the following can cause this error:

- Program too large
- Too many variables
- FOR loops nested more than 10 levels deep
- Subroutine calls nested more than 24 levels deep
- Parentheses nested more than 36 levels deep
- Too complicated an expression
- Attempt to set LOMEM: too high
- Attempt to set LOMEM: lower than present value
- Attempt to set HIMEM: too low

OVERFLOW [69]
The result of an arithmetic calculation was too large to be represented in Applesoft's internal number format.

REDIM'D ARRAY [120]
An attempt was made to define the same array twice in the same or different DIM statements. This error often occurs if an array has been referred to in a statement such as

LET A (I) = 3

before being defined in a DIM statement. At first reference, the array is automatically defined with an assumed dimension of 10; if such a
The statement is followed later in the program by

```
DIM A (100)
```

the REDIM'D ARRAY error will result. Another common cause of the error is a program that loops back to a line before the DIM statement, consequently executing it a second time.

This error message can prove useful if you wish to discover on what program line an array was defined: just insert a DIM statement for the array in the first line, run the program, and the program will halt with a REDIM'D ARRAY error when the original DIM statement is executed.

```
RETURN WITHOUT GOSUB [22]
```

A RETURN statement was encountered without a corresponding GOSUB having been executed. This error often occurs when control accidentally branches into a subroutine via a GOTO statement, or "falls into" a subroutine because there is no END or GOTO statement at the end of the program segment preceding the subroutine.

```
STRING TOO LONG [176]
```

An attempt was made by use of the concatenation operator (+) to create a string more than 255 characters long. This error tends to occur when a string variable is used more than once without being cleared (that is, without being reset to the null string).

```
SYNTAX ERROR [16]
```

A statement or expression doesn't conform to Applesoft's syntax rules. There are a myriad of possible causes for this error, such as a missing parenthesis, illegal character, or incorrect punctuation. Often results from a simple typing error.

```
TYPE MISMATCH [163]
```

The left side of an assignment statement was a numeric variable and the right side was a string, or vice versa; or a function that expected a string argument was given a numeric one or vice versa. Often caused by inadvertently leaving out the dollar sign ($) in a string variable or function name.
UNDEF'D FUNCTION [224]

A reference was made to a function that had never been defined. May occur when you type something like FN L(X) when you meant to type FN I(X); that is, a simple case of mistaken identifier.

UNDEF'D STATEMENT [90]

An attempt was made to transfer control, via GOTO, GOSUB, or IF...THEN, to a nonexistent line number. Common causes include accidentally deleting a line, changing a line number without changing references from other lines accordingly, and simple typing errors.
**Peeks, Pokes, and Calls**

This appendix discusses some of the many special features of the Apple IIe that you can use in your Applesoft programs by means of PEEK, POKE, or CALL statements. Notice that some of them duplicate the effects of other Applesoft features.

Many of these special addresses are **soft switches** with the property that any reference to them, whether a read (that is, a PEEK) or a write (a POKE), invokes the feature associated with the address. For instance, the example given here for switching from text to graphics without clearing the graphics screen is

\[ \text{POKE } -16304, 0 \]

but you can get the same effect by executing

\[ X = \text{PEEK} (-16304) \]

or by using POKE to address \(-16304\) with a value other than 0. This does not apply in cases where you must use POKE to store a specific value into the special address, such as a margin setting or a cursor location.

For more information on special features accessible with PEEK, POKE, and CALL, see the *Apple IIe Reference Manual*.

**Screen Text**

The special locations described in this section are used for controlling the display of text on the screen: setting the boundaries of the text window within which characters are displayed and scrolled, clearing all text from all or part of the screen, scrolling text within the text window, and controlling the position of the cursor.
Setting the **text window**

**Setting the text window** does not clear the remainder of the screen (for which you can use **HOME**) and does not move the cursor into the new text window (use **HOME** again, or **HTAB** and **VTAB**).

**POKE 32, L**

Sets the left edge of the text window to the value specified by expression **L**. This value should be between 0 and 39 (or 0 and 79 if you're using the 80-Column Text Card), where 0 represents the leftmost column of the screen.

The change doesn't become visible until the cursor attempts to return to the left edge of the window.

**Warning**

The width of the window is not changed by this statement: this means that the right edge will be moved by the same amount you move the left edge. To protect your program and Applesoft, first reduce the window width appropriately (see below); then change the left edge.

**POKE 33, W**

Sets the width of the text window (number of characters per line) to the value specified by expression **W**. This value should be between 1 and 40 (or 1 and 80 if you're using the 80-Column Text Card).

**Warning**

Make sure the right edge of the text window doesn't extend past the right edge of the display screen. The window width shouldn't be set greater than 40 (or 80) minus the current left edge of the window. For example, if you've set the left edge of the window (see above) to 10, don't set the window width greater than 30 (or 70 with the 80-Column Text Card).

Setting the window too wide will cause display text to be written outside the usual memory area reserved for it, destroying parts of your program or vital system information.

**Warning**

Do not set the window width to zero! The statement

```plaintext
POKE 33, 0
```

will cause the system to crash.

If **W** is less than 33, the **TAB** function in a **PRINT** statement may cause characters to be displayed outside the text window.

**Peeks, Pokes, and Calls**
POKE 34, T
Sets the top edge of the text window to the value specified by expression T. This value should be between 0 and 23, where 0 represents the top line of the screen.

Warning
Do not set the top edge of the window (T) lower than the bottom edge (see below).

POKE 35, B
Sets the bottom edge of the text window to the value specified by expression B. This value should be between 0 and 23, where 23 represents the bottom line of the screen.

Warning
Make sure the bottom of the text window doesn’t extend past the bottom of the display screen. Setting the window bottom beyond line 23 will cause display text to be written outside the usual memory area reserved for it, destroying parts of your program or vital system information.

Warning
Do not set the bottom edge of the window (B) higher than the top edge (see above).

CALL -936
Clears all text within the text window and moves the cursor to the top-left corner of the window. The effect is the same as that of the HOME statement or of typing [ESC] @ from the keyboard.

CALL -958
Clears all characters inside the text window from the current cursor position to the bottom-right corner. Characters above and to the left of the cursor are not affected. The effect is the same as that of typing [ESC] F from the keyboard.

CALL -868
Clears all characters inside the text window from the current cursor position to the end of the line. The effect is the same as that of typing [ESC] E from the keyboard.
CALL -922

Issues a line feed character, causing the cursor to move down one line without changing its horizontal position. If the cursor is on the bottom line of the text window, the contents of the window are scrolled up one line. The effect is the same as that of typing \[\text{CONTROL}\] -J from the keyboard.

CALL -912

Scrolls all text within the text window up one line. The old top line is lost; the old second line becomes the top line; the bottom line becomes blank. Text outside the text window is not affected.

PEEK (36)

Yields the current horizontal position of the cursor, which will be a number between 0 and 39 (0 and 79, if you're using the 80-Column Text Card). The cursor position is given relative to the left edge of the text window, not the left edge of the screen. The effect is the same as that of the POS function (see Section 5.2.4).

POKE 36, CH

Moves the cursor to the horizontal position specified by expression CH, which is interpreted relative to the left edge of the text window, not the left edge of the screen. The value of this expression should be between 0 and the current width of the window, with 0 representing the leftmost column of the window. The effect is the same as that of the HTAB statement (see Section 5.2.4), but is not limited to 40 columns.

Like HTAB, this statement can move the cursor beyond the right edge of the text window, but only long enough to display one character.

80-Column Text Card Users: You can use this POKE statement to position the cursor in columns 40 to 79 of the screen, which are inaccessible with HTAB.
Warning
Don't move the cursor past the right edge of the display screen! The cursor position shouldn't be set greater than 40 (or 80) minus the current left edge of the window. For example, if you've set the left edge of the window (see above) to 10, don't set the cursor position greater than 30 (or 70 with the 80-Column Text Card). Moving the cursor too far to the right may cause display text to be written outside the usual memory area reserved for it, destroying parts of your program or vital system information.

PEEK (37)
Yields the current vertical position of the cursor, which will be a number between 0 and 23. The cursor position is given relative to the top edge of the screen, not the top edge of the text window. A value of 0 represents the top line of the screen, 23 the bottom line.

POKE 37, CV
Moves the cursor to the vertical position specified by expression CV, which is interpreted relative to the top of the screen, not the top of the text window. The value of this expression should be between 0 and 23, with 0 representing the topmost line of the screen. The effect is similar to that of the VTAB statement (see Section 5.2.4), except that

- screen lines are numbered from 0 to 23, not from 1 to 24 as with VTAB
- the specified cursor position is not limited to 24 lines

Like VTAB, this statement can move the cursor beyond the bottom edge of the text window, but all subsequent text sent to the display screen will then appear on that same line.

Warning
Don't move the cursor past the bottom edge of the display screen! Setting the cursor position beyond line 23 will cause display text to be written outside the usual memory area reserved for it, destroying parts of your program or vital system information.
Keyboard

The special locations described below are used for reading input directly from the keyboard.

PEEK (-16384)

Reads the last character typed from the keyboard. If the high-order bit of this location is 1 (PEEK yields a result > 127), then a new character has been typed since the last POKE to address -16368 (see below); subtracting 128 from the value received gives the ASCII code for the character typed. If the high-order bit is 0 (PEEK yields a result ≥ 127), then no new character has been typed since the last POKE to -16368.

POKE -16368, 0

Clears the high-order bit of location -16384 (see above) to prepare for reading another keyboard character. This should be done immediately after reading the keyboard via PEEK (-16384).

Graphics

Four areas are reserved in the Apple IIe's memory for displaying text and graphics on the screen:

- Low-resolution page 1 is located at addresses $400 to $7FF hexadecimal (1024 to 2047 decimal). Information stored in this area can be interpreted and displayed on the screen in the form of either text or low-resolution graphics. This is the usual area of memory used for both these purposes, and is the area used by Applesoft's TEXT and GR statements.

- Low-resolution page 2, at addresses $800 to $BFF hexadecimal (2048 to 3071 decimal), can be used as an alternate area for either text or low-resolution graphics. Since this is the same area as the beginning of Applesoft's normal program storage space (see Section H.1, "Apple IIe Memory Map"), using it for text or graphics is tricky and is not recommended.

- High-resolution page 1, at addresses $2000 to $3FF hexadecimal (8192 to 16383 decimal), is the usual area for high-resolution graphics, and is accessible via Applesoft's HGR statement.

- High-resolution page 2, at addresses $4000 to $5FFF hexadecimal (16384 to 24575 decimal), serves as an alternate area for high-resolution graphics, and is accessible via Applesoft's HGR2 statement.

Peeks, Pokes, and Calls
**soft switch:** a location in memory that produces some special effect whenever its contents are read or written.

To use the different text and graphics areas, you can use Applesoft's built-in text and graphics facilities or you can use PEEK and POKE to manipulate the soft switches that control the display of text and graphics. There are four such soft switches, each consisting of a pair of special locations in the Apple Ile's memory. Any PEEK or POKE to one of the locations in the pair sets the switch one way; a PEEK or POKE to the other location in the pair sets the switch the other way.

The addresses shown in parentheses in the list below are those of the special locations that control the various settings of the switches. Each address is given first in hexadecimal (preceded by a dollar sign, $) and then in the equivalent decimal form.

The four soft switches controlling the display choose between

- text ($C051, -16303) and graphics ($C050, -16304)
- high ($C057, -16297) and low resolution ($C056, -16298)
- page 1 ($C054, -16300) and page 2 ($C055, -16299)
- full-screen graphics ($C052, -16302) and mixed text and graphics ($C053, -16301)

For more information...

POKE -16304, 0

Switches the display from full-screen text to graphics without clearing the graphics screen. Depending on the settings of the other soft switches, the resulting display may be high- or low-resolution graphics, taken from page 1 or 2, and full-screen graphics or mixed text and graphics.

**Similar Applesoft Statements:** The GR statement switches to mixed text and low-resolution graphics from page 1 and clears the graphics screen to black. The HGR statement switches to mixed text and high-resolution graphics from page 1 and clears the graphics screen to black. The HGR2 statement switches to full-screen high-resolution graphics from page 2 and clears the entire screen to black.
POKE -16303, 0

Displaying text

Switches the display from any form of graphics to full-screen text without resetting the text window. Depending on the setting of the applicable soft switch, the text displayed may be taken from low-resolution page 1 or page 2.

The TEXT statement also switches to text display, but in addition selects page 1, resets the text window to the full screen, and positions the cursor in the bottom-left corner of the screen (column 1, row 24).

POKE -16302, 0

Full-screen graphics

Switches the display from mixed text and graphics to full-screen graphics. Depending on the settings of the other soft switches, the resulting display may be either low- or high-resolution graphics and may be taken from either page 1 or page 2. If full-screen text is currently being displayed, there is no visible effect.

POKE -16301, 0

Mixed text and graphics

Switches the display from full-screen graphics to mixed text and graphics, with four lines of text at the bottom of the screen. Depending on the settings of the other soft switches, the upper portion of the screen may show low- or high-resolution graphics, taken from either page 1 or page 2. The text displayed in the bottom four lines will be taken from the same page number as the graphics in the upper part of the screen. If full-screen text is currently being displayed, there is no visible effect.

POKE -16300, 0

Displaying page 1

Switches the display from page 2 to page 1, without clearing the screen or moving the cursor. Depending on the settings of the other soft switches, the resulting display may be text or low-resolution graphics taken from low-resolution page 1, or high-resolution graphics taken from high-resolution page 1; if graphics, it may be either full-screen graphics or mixed with four lines of text from low-resolution page 1.

Always execute this POKE statement before switching to Integer BASIC if you've been using page 2 in Applesoft; otherwise you'll be left still looking at text (low-resolution) page 2 while Integer BASIC is writing its screen output to page 1.
POKE -16299, 0

Switches the display from page 1 to page 2, without clearing the screen or moving the cursor. Depending on the settings of the other soft switches, the resulting display may be text or low-resolution graphics taken from low-resolution page 2, or high-resolution graphics taken from high-resolution page 2; if graphics, it may be either full-screen graphics or mixed with four lines of text from low-resolution page 2.

POKE -16298, 0

Switches from high- to low-resolution graphics, without clearing the screen. Depending on the settings of the other soft switches, the resulting display may be taken from low-resolution page 1 or page 2, and may be either full-screen low-resolution graphics or mixed with four lines of text from the same low-resolution page. If full-screen text is currently being displayed, there is no visible effect.

Always execute this POKE statement before switching to Integer BASIC if you've been using high-resolution graphics in Applesoft; otherwise Integer BASIC's GR statement will incorrectly display the high- instead of the low-resolution page.

POKE -16297, 0

Switches from low- to high-resolution graphics, without clearing the screen. Depending on the settings of the other soft switches, the resulting display may be taken from high-resolution page 1 or page 2, and may be either full-screen high-resolution graphics or mixed with four lines of text from the corresponding low-resolution page. If full-screen text is currently being displayed, there is no visible effect.

CALL -1998

Clears low-resolution page 1 to black if displaying low-resolution graphics, or to black-on-white at-signs (@) if displaying text. If displaying high-resolution graphics, or text or low-resolution graphics from page 2, there is no visible effect.

CALL -1994

Clears the upper 40 rows of low-resolution page 1 to black if displaying low-resolution graphics, or the upper 20 lines to black-on-white at-signs (@) if displaying text. If displaying high-resolution graphics, or text or low-resolution graphics from page 2, there is no visible effect.
CALL -3086
Clears the current high-resolution page to black. (Applesoft remembers which page you used last, regardless of the switch settings.)

CALL -3082
Clears the current high-resolution page to the color most recently used in an HPLDT statement. (Applesoft remembers which page you used last, regardless of the switch settings.)

Miscellaneous Input and Output
This section describes the special locations in the Apple IIe's memory for controlling a variety of miscellaneous input and output devices: reading the buttons on the hand controls, controlling the annunciator outputs and the utility strobe, and producing sounds through the built-in speaker.

Reading the hand control buttons

The annunciators are four pins of the hand control connector that can each be set to either of two states (on or off). The utility strobe is another pin of the connector that is normally at +5 volts but can be triggered to drop to zero volts for one-half microsecond. These features are typically used to control devices such as lamps and relays connected to the computer through the hand control connector. See the Apple IIe Reference Manual for further information.

PEEK (-16287)
Reads the button on hand control 0; yields a result > 127 if the button is being pressed, <= 127 if it is not. The OPEN-APPLE key on the Apple IIe keyboard is equivalent to this button and can be read in the same way.

PEEK (-16286)
Reads the button on hand control 1; yields a result > 127 if the button is being pressed, <= 127 if it is not. The SOLID-APPLE key on the Apple IIe keyboard is equivalent to this button and can be read in the same way.
Controlling the annunciators

PEEK (-16285)
Reads the button on hand control 2; yields a result $> 127$ if the button is being pressed, $\leq 127$ if it is not.

Notice that, although there are provisions for connecting four hand controls (numbered 0 to 3) to the computer, there is no way to read the button on hand control 3.

POKE -16295, 0
Turns on annunciator output 0 (hand control connector, pin 15).

POKE -16296, 0
Turns off annunciator output 0 (hand control connector, pin 15).

POKE -16293, 0
Turns on annunciator output 1 (hand control connector, pin 14).

POKE -16294, 0
Turns off annunciator output 1 (hand control connector, pin 14).

POKE -16291, 0
Turns on annunciator output 2 (hand control connector, pin 13).

POKE -16292, 0
Turns off annunciator output 2 (hand control connector, pin 13).

POKE -16289, 0
Turns on annunciator output 3 (hand control connector, pin 12).

POKE -16290, 0
Turns off annunciator output 3 (hand control connector, pin 12).
Controlling the utility strobe

PEEK (-16320)
Triggers the utility strobe (hand control connector, pin 5).

The utility strobe should always be controlled with PEEK, not with POKE. Using POKE triggers the strobe twice instead of once. See the Apple Ile Reference Manual for further information.

Controlling the speaker

PEEK (-16336)
Produces a single click from the built-in speaker; can be used in various combinations and frequencies to produce musical tones and other sounds.

The speaker should always be controlled with PEEK, not with POKE. Using POKE produces two clicks instead of one. See the Apple Ile Reference Manual for further information.

PEEK (-16352)
Produces a single click on a cassette recording or on an audio amplifier connected to the cassette output jack via the amplifier's auxiliary input jack; can be used in various combinations and frequencies to produce musical tones and other sounds. See Appendix M for further information on using a cassette recorder.

Cassette output should always be controlled with PEEK, not with POKE. Using POKE produces two clicks instead of one. See the Apple Ile Reference Manual for further information.

Error Handling

This section describes the special locations associated with Applesoft's error handling mechanism. They can be used by user-supplied error-handling routines established with the ONERR GOTO statement. See Section 3.5 and Appendix E for further information.

PEEK (216)
Yields a result > 127 if an error-handling routine has been established with the ONERR GOTO statement, <= 127 if normal error handling is in effect.

ONERR GOTO statement: see Section 3.5.1

Peeks, Pokes, and Calls
POKE 216, 0
Restores Applesoft's normal error-handling mechanism; cancels the effect of a previous ON ERR GOTO statement.

PEEK (222)
After an error-handling routine has been called, yields the error code identifying the type of error detected. See Appendix E and Table 3-1 (Section 3.5.1) for further information on error codes.

Errors associated with the Disk Operating System (DOS) will also register at location 222; see the DOS manual for further information.

PEEK (219) * 256 + PEEK (218)
After an error-handling routine has been called, this expression yields the line number of the statement in which the error was detected.

CALL -3288
Clears from Applesoft's internal control stack information placed there when an error-handling routine was called. Should be used before exiting from any error-handling routine with a GOTO instead of a RESUME statement.

CALL 54915
Empties the internal control stack of all control information, without affecting the contents of any variables.
Hints for Program Efficiency

The information in this appendix can help you write programs that run faster or use less memory space. Section G.1, "Saving Space," gives tips you can follow if you need to conserve memory space. Section G.2, "Saving Time," suggests ways to speed up program execution.

Saving Space

Serious programmers often keep two versions of their programs: one expanded and heavily documented with REM statements, the other "crunched" to use the minimum memory space. There are a number of utility programs on the market that will make Applesoft programs more compact. They work by automatically removing REM statements, combining several statements onto a single program line, and eliminating optional semicolons in PRINT lists. Here are some tips for programmers who prefer to do the work themselves:

- Use multiple statements per line. There is a small amount of overhead (5 bytes) associated with each line in the program. Two of these bytes contain the line number. This means that no matter how many digits you have in your line number (minimum line number is 0, maximum is 63999), it takes the same number of bytes (two). Putting as many statements as possible on each line will cut down on the number of bytes used by your program. (A single line can include up to 239 characters.)

When combining statements into fewer lines, remember that when the condition in an IF...THEN statement is false, execution continues with the next line and not necessarily with the next statement.

If you're counting bytes, remember to add in one byte for each colon used to separate statements.
Combining many statements on one line makes editing and other changes much more difficult. It also makes a program more difficult to read and understand, not only for others but also for you yourself when you return to the program later on. Use this technique only in programs with serious space limitations.

- Delete all REM statements. Each REM statement uses at least one byte (for the keyword REM itself), plus one byte for each character in the text of the remark. For instance, the statement

  ```plaintext
  REM THIS IS A REMARK
  ```

  occupies 18 bytes of memory. In the program line

  ```plaintext
  140 X = X + Y : REM UPDATE SUM
  ```

  the REM uses 12 bytes of memory, plus one for the colon separating it from the preceding statement.

  Take care not to delete REM lines that are referred to by other lines. For example, if your program includes the lines

  ```plaintext
  200 GOTO 300
  .
  .
  300 REM THIS IS THE NEXT ROUTINE
  ```

  and you delete line 300, the program will halt with an UNDEF 'D STATEMENT error.

Like programs with many statements on each line, those without detailed REM statements are difficult to read and understand, not only for others but also for you yourself when you return to the program later on. You should consider eliminating REM statements only when faced with a serious shortage of memory space.

- Use integer instead of real arrays wherever possible (see Section H.2, "Applesoft Memory Allocation").

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

  ```plaintext
  PI = 3.14159
  ```

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

Hints for Program Efficiency
• Applesoft programs need not end with an END statement, so you can save a little space by omitting it.

• Reuse the same variables. If you use a variable T to hold a temporary result in one part of the program and you need a temporary variable later in your program, use T again. Or if you’re accepting a one-character input from the user in two different places in the program, use the same variable both times.

• Use subroutines and functions when the same action must be performed at different places in the program, to avoid having to write the identical code more than once.

• Use the zero elements of arrays—such as A(0) or B(0,X)—since space is allocated for them anyway.

• Semicolons are optional before and after TAB calls; leaving them out saves one byte per occurrence.

• Semicolons between items in a PRINT list are optional so long as the separate items are unambiguous. For instance, in line 50 of the following example all items will be displayed separately (although concatenated), since the dollar signs at the ends of the string variable names make it clear they are three separate variables:

```
10 LET A$ = "WELL,"
20 LET B$ = "MARSHA,"
30 LET C$ = "IT LOOKS LIKE"
40 LET C = 10
50 PRINT A$ B$ C$ "FRED IS "C" HOURS LATE!"
```

This program will display

```
WELL, MARSHA, IT LOOKS LIKE FRED IS 10 HOURS LATE!
```

But in this example several of the variables will be run together and interpreted as a single name:

```
10 LET A = 5
20 LET B = 10
30 LET C% = 15
40 LET D = 10
50 PRINT A B C% D
```
Applesoft interprets line 50 as saying "display the value of integer variable ABC, followed by the value of real variable D," and will display

010

Since variable ABC hasn't been assigned a value, its value is 0.

• If a quoted string is the last item in the last statement of a line, the closing quotation mark may be omitted, saving one byte:

10 PRINT "THIS IS THE WAY THE WORLD ENDS,
20 PRINT "NOT WITH A BANG BUT A WHIMPER

This last technique should be used with caution: bad things can happen if the omitted quotation mark comes somewhere other than at the end of a line:

10 PRINT "THIS WON'T WORK : PRINT "VERY WELL

This line will display

THIS WON'T WORK : PRINT 0

The final 0 is the value of the undefined variable VERYWELL.

Saving Time

Utility programs called compilers are now available that convert Applesoft programs to a form in which they run far faster than normally. However, a compiled program can take as much as 50% more space than a non-compiled one. The hints listed below should improve the execution speed of your Applesoft programs. Notice that some of these same hints were given in Section G.1 to save memory space. This means that in many cases you can both shorten and speed up your programs at the same time.

• This hint is probably ten times more important than any other in the list: use real variables wherever possible instead of integer variables or constants. It takes more time to convert an integer to its real-number representation than to fetch the value of a real variable. This technique is especially important within subroutines, loops, and other program segments that are executed repeatedly.
• Space for variables is allocated in the variable table in the order in which they are encountered during the execution of the program. A line such as

\[
5 \ A = 0 : B = A : C = B
\]

will place A first in the variable table, B second, and C third (assuming this line is the first executed in the program). When these variables are referred to later in the program, Applesoft will have to search only one entry in the variable table to find A, two entries to find B, and three entries to find C. Try to arrange for those variables that your program refers to most often to be located as early as possible in the variable table.

• Omit the index variable in NEXT statements. The statement

\[
\text{NEXT}
\]

is somewhat faster than

\[
\text{NEXT } I
\]

because no check needs to be made to see whether the variable named in the NEXT statement agrees with the index variable named in the corresponding FOR statement.

• When Applesoft encounters a backward reference from a later line of the program to an earlier one, such as

\[
900 \ \text{GOTO} \ 100
\]

it scans the entire program from the beginning until it finds the desired line number (100, in this example). So you can speed things up by placing frequently-referenced lines as early in the program as possible.
This appendix contains information on various details of Applesoft's internal operation:

- Section H.1, "Apple Ile Memory Map," summarizes the use of memory in the Apple Ile and identifies areas of memory reserved for system use.
- Section H.2, "Applesoft Memory Allocation," describes the way Applesoft allocates memory for program and variable storage.
- Section H.4, "Keyword Tokens," lists the internal codes Applesoft uses to represent keywords occurring in a program.
# Apple Ile Memory Map

Table H-1 summarizes memory usage in the Apple Ile. Addresses preceded by a dollar sign ($) are in hexadecimal; they are followed on the next line by their decimal equivalents.

<table>
<thead>
<tr>
<th>Memory Range</th>
<th>From</th>
<th>To</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0000</td>
<td>$01FF</td>
<td>System workspace; not advisable to use</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>511</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0200</td>
<td>$02FF</td>
<td>Keyboard character buffer</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>767</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0300</td>
<td>$03CF</td>
<td>Available for short machine-language programs</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>975</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$03D0</td>
<td>$03FF</td>
<td>Used by DOS (available if you don’t use a disk drive)</td>
</tr>
<tr>
<td></td>
<td>976</td>
<td>1023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0400</td>
<td>$07FF</td>
<td>Low-resolution graphics and text display, page 1</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>2047</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0800</td>
<td>$0BFF</td>
<td>Low-resolution graphics and text display, page 2</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>3071</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0800</td>
<td>$0XXX</td>
<td>Applesoft program and variable space, where XXX is the setting of HIMEM1. This may be set as high as 49151; must be less if using DOS or reserving part of memory for machine language routines or high-resolution display pages.</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>XXX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$2000</td>
<td>$3FFF</td>
<td>High-resolution graphics display page 1</td>
</tr>
<tr>
<td></td>
<td>8192</td>
<td>16383</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$4000</td>
<td>$5FFF</td>
<td>High-resolution graphics display page 2</td>
</tr>
<tr>
<td></td>
<td>16384</td>
<td>24575</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$9600</td>
<td>$BFFF</td>
<td>DOS (Disk Operating System)</td>
</tr>
<tr>
<td></td>
<td>38400</td>
<td>49151</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C000</td>
<td>$CFFF</td>
<td>Hardware I/O Addresses</td>
</tr>
<tr>
<td></td>
<td>49152</td>
<td>53247</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D000</td>
<td>$F7FF</td>
<td>Applesoft</td>
</tr>
<tr>
<td></td>
<td>53248</td>
<td>63487</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$F800</td>
<td>$FFFF</td>
<td>Apple Ile System Monitor</td>
</tr>
<tr>
<td></td>
<td>63488</td>
<td>65535</td>
<td></td>
</tr>
</tbody>
</table>
**Applesoft Memory Allocation**

Figure H-1 shows how Applesoft allocates the memory space between the start of program storage (normally $800$ hexadecimal, 2048 decimal) and the end of variable storage (determined by the setting of HIMEM:). The boundaries between areas may vary; the left column gives "pointer" addresses at which the current settings of the boundaries can be found.

<table>
<thead>
<tr>
<th>Area</th>
<th>Start Address</th>
<th>End Address</th>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM</td>
<td>$0067-$006B</td>
<td>103-104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPLE VARIABLES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARRAYS</td>
<td>$006B-$006C</td>
<td>107-108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREE SPACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRINGS</td>
<td>$006F-$0070</td>
<td>111-112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>end of strings (HIMEM:)</td>
<td>$0073-$0074</td>
<td>115-116</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointer addresses are given in hexadecimal first, followed by their decimal equivalents. All pointers are stored with the low-order byte first. Thus, for example, the address of the beginning of string space can be calculated with the Applesoft expression

$$\text{PEEK}(112) \times 256 + \text{PEEK}(111)$$
Figure H-2 shows how space is allocated for individual variables and arrays. Simple real, integer, or string variables use seven bytes each. Real variables use two bytes for the variable name and five for the value (one exponent, four mantissa, most significant first). Integer variables use two bytes for the variable name, two for the value, and have zeros in the remaining three bytes. String variables use two bytes for the variable name, one for the length of the string, two for a pointer to the contents of the string in memory, and have zeros in the remaining two bytes.

Real arrays use a minimum of twelve bytes: two bytes for the array name, two for the size of the array in bytes, one for the number of dimensions, two for each dimension, and five for each element of the array. Integer array variables use only two bytes for each element. String array variables use three bytes for each element: one for the length of the string and two for a pointer to its contents. Multidimensional arrays are stored with the first subscript varying fastest.

String variables and arrays contain pointers (addresses) to the contents of the strings themselves, which are stored in order of creation from HIMEM: down, using one byte of memory for each character in the string. The pointer stored with the variable gives the address of the first character in the string.

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

Reserved words occurring in a program are converted into one-byte tokens (see Section H.4, "Keyword Tokens"). All other characters in a program occupy one byte of program storage each.

As a program is executed, space is allocated on the internal control stack as follows:

- Each active FOR/NEXT loop uses 16 bytes.
- Each active subroutine (one that has been called and has not yet returned) uses 6 bytes.
- Each pair of parentheses encountered in an expression uses 4 bytes.
- Each temporary result calculated in an expression uses 12 bytes.
### Variable and Array Maps

#### Simple Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>NAME (pos)</th>
<th>(neg) 1st byte</th>
<th>(neg) 2nd byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>exponent</td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mantissa</td>
<td>m.s. byte</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>exponent</td>
<td>m.s. byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mantissa</td>
<td>m.s. byte</td>
<td></td>
</tr>
<tr>
<td>String Pointers</td>
<td>NAME (pos)</td>
<td>(neg) 1st byte</td>
<td>(neg) 2nd byte</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>address</td>
<td>low byte</td>
<td>high byte</td>
</tr>
</tbody>
</table>

#### Array Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>NAME (pos)</th>
<th>(neg) 1st byte</th>
<th>(neg) 2nd byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>OFFSET</td>
<td>pointer to next variable: add to address of this variable name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NO. OF DIMENSIONS</td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE Nth DIMENSION</td>
<td>high byte</td>
<td>low byte</td>
</tr>
<tr>
<td></td>
<td>SIZE 1st DIMENSION</td>
<td>high byte</td>
<td>low byte</td>
</tr>
<tr>
<td>Integer</td>
<td>OFFSET</td>
<td>pointer to next variable: add to address of this variable name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NO. OF DIMENSIONS</td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE Nth DIMENSION</td>
<td>high byte</td>
<td>low byte</td>
</tr>
<tr>
<td></td>
<td>SIZE 1st DIMENSION</td>
<td>high byte</td>
<td>low byte</td>
</tr>
<tr>
<td>String Pointers</td>
<td>OFFSET</td>
<td>pointer to next variable: add to address of this variable name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NO. OF DIMENSIONS</td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE Nth DIMENSION</td>
<td>high byte</td>
<td>low byte</td>
</tr>
<tr>
<td></td>
<td>SIZE 1st DIMENSION</td>
<td>high byte</td>
<td>low byte</td>
</tr>
</tbody>
</table>

### Examples

- **REAL** $(0, 0, \ldots, 0)$
  - exponent: 1 byte
  - mantissa: m.s. byte
  - mantissa: l.s. byte

- **REAL** $(N, N, \ldots, N)$
  - exponent: 1 byte
  - mantissa: m.s. byte
  - mantissa: l.s. byte

- **INTEGER%** $(0, 0, \ldots, 0)$
  - high byte
  - low byte

- **STRING$** $(0, 0, \ldots, 0)$
  - length: 1 byte
  - address: high byte

---

m. s. = most significant
l. s. = least significant

**Applesoft Memory Location**
Zero Page Usage

Table H-2 shows the locations that Applesoft uses in page zero of memory (locations $0000$ through $00FF$ hexadecimal). Addresses are given first in hexadecimal, then in decimal. All pointers (memory addresses) are stored in the usual 6502 style, low-order byte first. To find the value of a pointer, use the Applesoft expression

\[
\text{PEEK (SECONDADDR) * 256 + PEEK (FIRSTADDR)}
\]

where \text{FIRSTADDR} and \text{SECONDADDR} are the addresses of the two bytes of the pointer itself.

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000 - 0005$</td>
<td>Jump instructions to continue in Applesoft</td>
</tr>
<tr>
<td>0 - 5</td>
<td></td>
</tr>
<tr>
<td>$000A - 000C$</td>
<td>Jump instruction for USR function (see Section 7.1.4)</td>
</tr>
<tr>
<td>10 - 12</td>
<td></td>
</tr>
<tr>
<td>$000D - 0017$</td>
<td>General purpose counters/flags for Applesoft</td>
</tr>
<tr>
<td>13 - 23</td>
<td></td>
</tr>
<tr>
<td>$0020 - 004F$</td>
<td>Reserved for system Monitor program</td>
</tr>
<tr>
<td>32 - 79</td>
<td></td>
</tr>
<tr>
<td>$0050 - 0061$</td>
<td>General purpose pointers for Applesoft</td>
</tr>
<tr>
<td>80 - 97</td>
<td></td>
</tr>
<tr>
<td>$0062 - 0066$</td>
<td>Result of last multiply or divide</td>
</tr>
<tr>
<td>98 - 102</td>
<td></td>
</tr>
<tr>
<td>$0067 - 0068$</td>
<td>Pointer to beginning of program. Normally set to $0801$.</td>
</tr>
<tr>
<td>103 - 104</td>
<td></td>
</tr>
<tr>
<td>$0069 - 006A$</td>
<td>Pointer to start of simple variable space. Also points to the end of the program plus 1 or 2, unless changed with the \text{LOMEM}: statement.</td>
</tr>
<tr>
<td>105 - 106</td>
<td></td>
</tr>
<tr>
<td>$006B - 006C$</td>
<td>Pointer to start of array space</td>
</tr>
<tr>
<td>107 - 108</td>
<td></td>
</tr>
<tr>
<td>$006D - 006E$</td>
<td>Pointer to end of array space</td>
</tr>
<tr>
<td>109 - 110</td>
<td></td>
</tr>
<tr>
<td>$006F - 0070$</td>
<td>Pointer to start of string start. Strings are stored from here to value of \text{HIMEM}:</td>
</tr>
<tr>
<td>111 - 112</td>
<td></td>
</tr>
<tr>
<td>$0071 - 0072$</td>
<td>General pointer</td>
</tr>
<tr>
<td>113 - 114</td>
<td></td>
</tr>
<tr>
<td>$0073 - 0074$</td>
<td>Highest location in memory available to Applesoft plus one. On initial entry to Applesoft, set to the highest RAM memory location available.</td>
</tr>
<tr>
<td>115 - 116</td>
<td></td>
</tr>
<tr>
<td>Location(s)</td>
<td>Used for</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>$0075 - $0076</td>
<td>Line number of line currently being executed</td>
</tr>
<tr>
<td>117 - 118</td>
<td></td>
</tr>
<tr>
<td>$0077 - $0078</td>
<td>&quot;Old line number&quot; at which execution was interrupted by</td>
</tr>
<tr>
<td>119 - 120</td>
<td>END, STOP, or C</td>
</tr>
<tr>
<td>$0079 - $007A</td>
<td>&quot;Old text pointer.&quot; Location in memory of statement to be</td>
</tr>
<tr>
<td>121 - 122</td>
<td>executed next</td>
</tr>
<tr>
<td>$007B - $007C</td>
<td>Line number of DATA statement containing next item for</td>
</tr>
<tr>
<td>123 - 124</td>
<td>READ</td>
</tr>
<tr>
<td>$007D - $007E</td>
<td>Absolute memory location of next item for READ</td>
</tr>
<tr>
<td>125 - 126</td>
<td></td>
</tr>
<tr>
<td>$007F - $0080</td>
<td>Pointer to current source of INPUT. Set to $0201 during</td>
</tr>
<tr>
<td>127 - 128</td>
<td>an INPUT statement. During a READ statement, set to the</td>
</tr>
<tr>
<td></td>
<td>DATA item being read.</td>
</tr>
<tr>
<td>$0081 - $0082</td>
<td>Name of last-used variable</td>
</tr>
<tr>
<td>129 - 130</td>
<td></td>
</tr>
<tr>
<td>$0083 - $0084</td>
<td>Pointer to value of last-used variable</td>
</tr>
<tr>
<td>131 - 132</td>
<td></td>
</tr>
<tr>
<td>$0085 - $008C</td>
<td>General use</td>
</tr>
<tr>
<td>133 - 156</td>
<td></td>
</tr>
<tr>
<td>$008D - $00A3</td>
<td>Main floating-point accumulator</td>
</tr>
<tr>
<td>157 - 163</td>
<td></td>
</tr>
<tr>
<td>$00A4</td>
<td>General use in floating-point arithmetic</td>
</tr>
<tr>
<td>164</td>
<td></td>
</tr>
<tr>
<td>$00A5 - $00AB</td>
<td>Secondary floating-point accumulator</td>
</tr>
<tr>
<td>165 - 171</td>
<td></td>
</tr>
<tr>
<td>$00AC - $00AE</td>
<td>General use flags/pointers</td>
</tr>
<tr>
<td>172 - 174</td>
<td></td>
</tr>
<tr>
<td>$00AF - $00B0</td>
<td>Pointer to end of program (not changed by LOMEM :)</td>
</tr>
<tr>
<td>175 - 176</td>
<td></td>
</tr>
<tr>
<td>$00B1 - $00CB</td>
<td>Character input routine. Applesoft calls here every time it</td>
</tr>
<tr>
<td>177 - 200</td>
<td>wants another character.</td>
</tr>
<tr>
<td>$00B8 - $00B9</td>
<td>Pointer to last character obtained through character input</td>
</tr>
<tr>
<td>184 - 185</td>
<td>routine</td>
</tr>
<tr>
<td>$00C9 - $00CD</td>
<td>Random number</td>
</tr>
<tr>
<td>201 - 205</td>
<td></td>
</tr>
<tr>
<td>$00D0 - $00D5</td>
<td>High-resolution graphics scratch pointers</td>
</tr>
<tr>
<td>208 - 213</td>
<td></td>
</tr>
<tr>
<td>$00DB - $00DF</td>
<td>DNERF pointers/scratch</td>
</tr>
<tr>
<td>216 - 223</td>
<td></td>
</tr>
<tr>
<td>$00E0 - $00E2</td>
<td>High-resolution graphics horizontal and vertical coordinates</td>
</tr>
<tr>
<td>224 - 226</td>
<td></td>
</tr>
</tbody>
</table>
### Table H-2 continued

<table>
<thead>
<tr>
<th>Location(s)</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00E4 228</td>
<td>High-resolution graphics color code</td>
</tr>
<tr>
<td>$00E5 - $00E7 229 - 231</td>
<td>General use for high-resolution graphics</td>
</tr>
<tr>
<td>$00E6 230</td>
<td>Current high-resolution page being drawn on (decimal 32 if page 1; decimal 64 if page 2)</td>
</tr>
<tr>
<td>$00E8 - $00E9 232 - 233</td>
<td>Pointer to beginning of shape table</td>
</tr>
<tr>
<td>$00F0 - $00F3 240 - 243</td>
<td>General use flags</td>
</tr>
<tr>
<td>$00F4 - $00FB 244 - 248</td>
<td>ONERR pointers</td>
</tr>
</tbody>
</table>

### Keyword Tokens

Applesoft *tokenizes* all its key words; that is, it converts them to one-byte codes called *tokens* to save memory space. Table H-3 gives a list of the tokens representing the various keywords.

### Table H-3 Applesoft Keyword Tokens

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Keyword</th>
<th>Hex</th>
<th>Dec</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$80</td>
<td>128</td>
<td>END</td>
<td>$8E</td>
<td>142</td>
<td>HLIN</td>
</tr>
<tr>
<td>$81</td>
<td>129</td>
<td>FDR</td>
<td>$8F</td>
<td>143</td>
<td>VLIN</td>
</tr>
<tr>
<td>$82</td>
<td>130</td>
<td>NEXT</td>
<td>$90</td>
<td>144</td>
<td>HGR2</td>
</tr>
<tr>
<td>$83</td>
<td>131</td>
<td>DATA</td>
<td>$91</td>
<td>145</td>
<td>HGR</td>
</tr>
<tr>
<td>$84</td>
<td>132</td>
<td>INPUT</td>
<td>$92</td>
<td>146</td>
<td>HCOLOR</td>
</tr>
<tr>
<td>$85</td>
<td>133</td>
<td>DEL</td>
<td>$93</td>
<td>147</td>
<td>HCOLOR =</td>
</tr>
<tr>
<td>$86</td>
<td>134</td>
<td>DIM</td>
<td>$94</td>
<td>148</td>
<td>DRAW</td>
</tr>
<tr>
<td>$87</td>
<td>135</td>
<td>READ</td>
<td>$95</td>
<td>149</td>
<td>XDRAW</td>
</tr>
<tr>
<td>$88</td>
<td>136</td>
<td>GR</td>
<td>$96</td>
<td>150</td>
<td>HTAB</td>
</tr>
<tr>
<td>$89</td>
<td>137</td>
<td>TEXT</td>
<td>$97</td>
<td>151</td>
<td>HOME</td>
</tr>
<tr>
<td>$8A</td>
<td>138</td>
<td>PR#</td>
<td>$98</td>
<td>152</td>
<td>ROT</td>
</tr>
<tr>
<td>$8B</td>
<td>139</td>
<td>IN#</td>
<td>$99</td>
<td>153</td>
<td>SCALE</td>
</tr>
<tr>
<td>$8C</td>
<td>140</td>
<td>CALL</td>
<td>$9A</td>
<td>154</td>
<td>SHLOAD</td>
</tr>
<tr>
<td>$8D</td>
<td>141</td>
<td>PLOT</td>
<td>$9B</td>
<td>155</td>
<td>TRACE</td>
</tr>
<tr>
<td>Hex</td>
<td>Dec</td>
<td>Keyword</td>
<td>Hex</td>
<td>Dec</td>
<td>Keyword</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>---------</td>
<td>-------</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>$8C</td>
<td>156</td>
<td>NOTRACE</td>
<td>$BB</td>
<td>187</td>
<td>CONT</td>
</tr>
<tr>
<td>$8D</td>
<td>157</td>
<td>NORMAL</td>
<td>$BC</td>
<td>188</td>
<td>LIST</td>
</tr>
<tr>
<td>$8E</td>
<td>158</td>
<td>INVERSE</td>
<td>$BD</td>
<td>189</td>
<td>CLEAR</td>
</tr>
<tr>
<td>$8F</td>
<td>159</td>
<td>FLASH</td>
<td>$BE</td>
<td>190</td>
<td>GET</td>
</tr>
<tr>
<td>$A0</td>
<td>160</td>
<td>COLOR=</td>
<td>$BF</td>
<td>191</td>
<td>NEW</td>
</tr>
<tr>
<td>$A1</td>
<td>161</td>
<td>POP</td>
<td>$C0</td>
<td>192</td>
<td>TAB</td>
</tr>
<tr>
<td>$A2</td>
<td>162</td>
<td>VTABLE</td>
<td>$C1</td>
<td>193</td>
<td>TO</td>
</tr>
<tr>
<td>$A3</td>
<td>163</td>
<td>HIMEM:</td>
<td>$C2</td>
<td>194</td>
<td>FN</td>
</tr>
<tr>
<td>$A4</td>
<td>164</td>
<td>LOMEM:</td>
<td>$C3</td>
<td>195</td>
<td>SPC</td>
</tr>
<tr>
<td>$A5</td>
<td>165</td>
<td>ONERR</td>
<td>$C4</td>
<td>196</td>
<td>THEN</td>
</tr>
<tr>
<td>$A6</td>
<td>166</td>
<td>RESUME</td>
<td>$C5</td>
<td>197</td>
<td>AT</td>
</tr>
<tr>
<td>$A7</td>
<td>167</td>
<td>RECALL</td>
<td>$C6</td>
<td>198</td>
<td>NOT</td>
</tr>
<tr>
<td>$A8</td>
<td>168</td>
<td>STORE</td>
<td>$C7</td>
<td>199</td>
<td>STEP</td>
</tr>
<tr>
<td>$A9</td>
<td>169</td>
<td>SPEED=</td>
<td>$C8</td>
<td>200</td>
<td>+</td>
</tr>
<tr>
<td>$AA</td>
<td>170</td>
<td>LET</td>
<td>$C9</td>
<td>201</td>
<td>-</td>
</tr>
<tr>
<td>$AB</td>
<td>171</td>
<td>GOTO</td>
<td>$CA</td>
<td>202</td>
<td>*</td>
</tr>
<tr>
<td>$AC</td>
<td>172</td>
<td>RUN</td>
<td>$CB</td>
<td>203</td>
<td>/</td>
</tr>
<tr>
<td>$AD</td>
<td>173</td>
<td>IF</td>
<td>$CC</td>
<td>204</td>
<td>^</td>
</tr>
<tr>
<td>$AE</td>
<td>174</td>
<td>RESTORE</td>
<td>$CD</td>
<td>205</td>
<td>AND</td>
</tr>
<tr>
<td>$AF</td>
<td>175</td>
<td>&amp;</td>
<td>$CE</td>
<td>206</td>
<td>OR</td>
</tr>
<tr>
<td>$B0</td>
<td>176</td>
<td>GOSUB</td>
<td>$CF</td>
<td>207</td>
<td>&gt;</td>
</tr>
<tr>
<td>$B1</td>
<td>177</td>
<td>RETURN</td>
<td>$D0</td>
<td>208</td>
<td>=</td>
</tr>
<tr>
<td>$B2</td>
<td>178</td>
<td>REM</td>
<td>$D1</td>
<td>209</td>
<td>&lt;</td>
</tr>
<tr>
<td>$B3</td>
<td>179</td>
<td>STDP</td>
<td>$D2</td>
<td>210</td>
<td>SGN</td>
</tr>
<tr>
<td>$B4</td>
<td>180</td>
<td>ON</td>
<td>$D3</td>
<td>211</td>
<td>INT</td>
</tr>
<tr>
<td>$B5</td>
<td>181</td>
<td>WAIT</td>
<td>$D4</td>
<td>212</td>
<td>ABS</td>
</tr>
<tr>
<td>$B6</td>
<td>182</td>
<td>LOAD</td>
<td>$D5</td>
<td>213</td>
<td>USR</td>
</tr>
<tr>
<td>$B7</td>
<td>183</td>
<td>SAVE</td>
<td>$D6</td>
<td>214</td>
<td>FRE</td>
</tr>
<tr>
<td>$B8</td>
<td>184</td>
<td>DEF</td>
<td>$D7</td>
<td>215</td>
<td>SCRN</td>
</tr>
<tr>
<td>$B9</td>
<td>185</td>
<td>POKE</td>
<td>$D8</td>
<td>216</td>
<td>PDL</td>
</tr>
<tr>
<td>$BA</td>
<td>186</td>
<td>PRINT</td>
<td>$D9</td>
<td>217</td>
<td>POS</td>
</tr>
</tbody>
</table>

Implementation Details
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Keyword</th>
<th>Hex</th>
<th>Dec</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DA</td>
<td>218</td>
<td>SQR</td>
<td>$E3</td>
<td>227</td>
<td>LEN</td>
</tr>
<tr>
<td>$DB</td>
<td>219</td>
<td>RND</td>
<td>$E4</td>
<td>228</td>
<td>STR$</td>
</tr>
<tr>
<td>$DC</td>
<td>220</td>
<td>LOG</td>
<td>$E5</td>
<td>229</td>
<td>VAL</td>
</tr>
<tr>
<td>$DD</td>
<td>221</td>
<td>EXP</td>
<td>$E6</td>
<td>230</td>
<td>ASC</td>
</tr>
<tr>
<td>$DE</td>
<td>222</td>
<td>COS</td>
<td>$E7</td>
<td>231</td>
<td>CHR$</td>
</tr>
<tr>
<td>$DF</td>
<td>223</td>
<td>SIN</td>
<td>$E8</td>
<td>232</td>
<td>LEFT$</td>
</tr>
<tr>
<td>$E0</td>
<td>224</td>
<td>TAN</td>
<td>$E9</td>
<td>233</td>
<td>RIGHT$</td>
</tr>
<tr>
<td>$E1</td>
<td>225</td>
<td>ATN</td>
<td>$EA</td>
<td>234</td>
<td>MID$</td>
</tr>
<tr>
<td>$E2</td>
<td>226</td>
<td>PEEK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Display Formats for Numbers

This appendix describes the formats in which Applesoft displays or prints numeric values. Numbers may not always be formatted in the way you might expect; this is particularly true for numbers more than 9 digits long or for exceptionally small numbers.

Numeric values in Applesoft must be in the range $-1 \times 10^{38}$ to $1 \times 10^{38}$. Any number whose absolute value is less than approximately $3 \times 10^{-39}$ is converted to zero. True integer values to be assigned to integer variables (such as `A%`) must be in the range $-32767$ to $+32767$.

A number typed from the keyboard or a numeric constant used in an Applesoft program may have as many as 38 digits. However, only nine digits are significant, and the last digit is rounded off. An Applesoft statement that you type as

```
PRINT 1.23456787654321
```

—you type this from the keyboard

will display

```
1.23456788
```

—you get this on the screen

on the screen.

All arithmetic done on reals

`truncate`: to convert a real number to the next lowest integer

Integers are always converted to real form before being used in arithmetic calculations, and the results are converted back to integer form when assigned to an integer variable. Conversion from real to integer form is by truncation to the next lowest integer, not by rounding to the nearest integer.
Rules for number formats

Applesoft displays and prints numbers according to the following rules:

- If the number is negative, it is preceded by a minus sign (-); if it is zero or positive, no sign is used.
- If the number is an integer with an absolute value from 0 to 999,999,999, it is formatted as an integer.
- If the number is not an integer and its absolute value is between 0.01 and 999,999,999.2, it is formatted with a decimal point in the usual way.
- In all other cases, the number is formatted in scientific notation (see below).

Table 1-1 shows examples of the formats used for displaying and printing numbers.

<table>
<thead>
<tr>
<th>Number</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>6523</td>
<td>6523</td>
</tr>
<tr>
<td>-23.460</td>
<td>-23.46</td>
</tr>
<tr>
<td>45.72 * 10^5</td>
<td>4572000</td>
</tr>
<tr>
<td>1 * 10^20</td>
<td>1E+20</td>
</tr>
<tr>
<td>-12.34567896 * 10^-10</td>
<td>-1.2345679E+11</td>
</tr>
<tr>
<td>1000000000</td>
<td>1E+09</td>
</tr>
<tr>
<td>9999999999</td>
<td>9999999999</td>
</tr>
</tbody>
</table>
The format Applesoft uses for scientific notation is shown in Figure I-1. A sign is shown only if the number is negative. There is always exactly one nonzero digit before the decimal point and up to eight digits after it, with trailing zeros suppressed. There are never any leading zeros; the digit before the decimal point is always nonzero. If there is only one digit to print after all trailing zeros are suppressed, no decimal point is shown. The letter E (for "exponent") is always followed by a sign and a two-digit exponent. The value of a number represented in this form is the number before the E times 10 raised to the power after the E. For example,

\[
\begin{align*}
\text{PRINT } & 35 \times 345^{14} \quad \text{yields } 1.18450085E+37 \\
\text{PRINT } & -3.14159 \times 567^{5} \quad \text{yields } -1.84104669E+14 \\
\text{PRINT } & 1 / 999 \quad \text{yields } 1.001001E-03 \\
\text{PRINT } & -3 / 999 \quad \text{yields } -3.003003E-03
\end{align*}
\]

Display Formats for Numbers
On-Screen Editing and Cursor Control

The figures and tables below summarize Applesoft's facilities for on-screen editing and cursor control. These features are discussed briefly in Section 1.4, "Editing What You Type," and at greater length in the *Apple IIe Owner's Manual* and the *Apple IIe Applesoft Tutorial*.

If you have the Apple IIe 80-Column Text Card installed in your computer, additional escape-mode features are available; see the *80-Column Text Card Manual*.

**Figure J-1** Single Cursor Moves

![Single Cursor Moves Diagram]

**Figure J-2** Long-range Cursor Moves

![Long-range Cursor Moves Diagram]
**Table J-1** ASCII Equivalents of Arrow Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>ASCII Code</th>
<th>Keyboard Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT-ARROW</td>
<td>8</td>
<td>CONTROL-H</td>
</tr>
<tr>
<td>RIGHT-ARROW</td>
<td>21</td>
<td>CONTROL-U</td>
</tr>
<tr>
<td>UP-ARROW</td>
<td>11</td>
<td>CONTROL-K</td>
</tr>
<tr>
<td>DOWN-ARROW</td>
<td>10</td>
<td>CONTROL-J</td>
</tr>
</tbody>
</table>

**Table J-2** Escape-Mode Functions

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Moves cursor right one position; leaves escape mode</td>
</tr>
<tr>
<td>B</td>
<td>Moves cursor left one position; leaves escape mode</td>
</tr>
<tr>
<td>C</td>
<td>Moves cursor down one line; leaves escape mode</td>
</tr>
<tr>
<td>D</td>
<td>Moves cursor up one line; leaves escape mode</td>
</tr>
<tr>
<td>I</td>
<td>Moves cursor up one line; remains in escape mode</td>
</tr>
<tr>
<td>J</td>
<td>Moves cursor left one position; remains in escape mode</td>
</tr>
<tr>
<td>K</td>
<td>Moves cursor right one position; remains in escape mode</td>
</tr>
<tr>
<td>M</td>
<td>Moves cursor down one line; remains in escape mode</td>
</tr>
<tr>
<td>LEFT-ARROW</td>
<td>Moves cursor left one position; remains in escape mode</td>
</tr>
<tr>
<td>RIGHT-ARROW</td>
<td>Moves cursor right one position; remains in escape mode</td>
</tr>
<tr>
<td>UP-ARROW</td>
<td>Moves cursor up one line; remains in escape mode</td>
</tr>
<tr>
<td>DOWN-ARROW</td>
<td>Moves cursor down one line; remains in escape mode</td>
</tr>
<tr>
<td>E</td>
<td>Clears from cursor to end of line; leaves escape mode</td>
</tr>
<tr>
<td>F</td>
<td>Clears from cursor to end of text window; leaves escape mode</td>
</tr>
<tr>
<td>@</td>
<td>Clears entire text window; moves cursor to top-left corner; leaves escape mode</td>
</tr>
</tbody>
</table>
The following chart summarizes the differences in the Apple IIe's behavior with and without the Apple IIe 80-Column Text Card installed. Notice that even 40-column display behaves somewhat differently with the 80-Column Text Card installed and active than without it. See the 80-Column Text Card Manual for further information.

<table>
<thead>
<tr>
<th>Table K-1 40/80-Column Display Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Escape Mode</strong></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td><strong>Card Inactive:</strong></td>
</tr>
<tr>
<td><strong>40-Column Display</strong></td>
</tr>
<tr>
<td><strong>Card Active:</strong></td>
</tr>
<tr>
<td><strong>40-Column Display</strong></td>
</tr>
<tr>
<td><strong>Card Active:</strong></td>
</tr>
<tr>
<td><strong>80-Column Display</strong></td>
</tr>
<tr>
<td><strong>Card Active:</strong></td>
</tr>
<tr>
<td><strong>40-Column Display</strong></td>
</tr>
</tbody>
</table>
Comparison with Integer BASIC

This appendix summarizes the differences between Applesoft and Apple's earlier Integer BASIC language. Section L.3 gives some hints on converting programs written in Integer or other versions of BASIC to Applesoft.

If Integer BASIC is loaded into your computer's memory, you can switch from Applesoft to Integer BASIC by typing the command

INT

To switch from Integer BASIC to Applesoft, type

FP

FP stands for "floating-point," the name for the internal format used by languages like Applesoft to represent real numbers.
## Differences between Features

Table L-1 lists Applesoft statements and functions that are not available in Integer BASIC.

| ATN | CHR$ | COS | DATA | DEF | FN | DRAW | EXP | FLASH | FN | FRE | GET | HCOLOR = HGR | HGR2 | HIMEM: | HOME | HPLT | INT | INVERSE | LEFT$ | LOG | LOMEM: | MID$ | NORMAL | ON...GOSUB | ON...GOTO | ONERR | GOTO | POS | READ | RECALL | RESTORE | RESUME | RIGHT$ | ROT = SCALE | SHLOAD | SIN | SPC | SPEED | SQRT | STOP | STORE | STR$ | TAN | USR | VAL | WAIT | XDRAW |
|-----|------|-----|------|-----|----|------|-----|-------|----|-----|-----|----------|------|---------|-------|--------|-----|---------|------|-----|--------|-----|--------|----------|----------|-------|--------|-----|-------|------|-------|-------|--------|-----|-----|-----|------|

Table L-2 lists Integer BASIC statements and functions that are not available in Applesoft.

| AUTO | DSP | MAN | MOD |
Table L-3 lists Applesoft features that are expressed or accomplished differently in Integer BASIC.

<table>
<thead>
<tr>
<th>Applesoft</th>
<th>Integer BASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>CLR</td>
</tr>
<tr>
<td>CONT</td>
<td>CON</td>
</tr>
<tr>
<td>HTAB</td>
<td>TAB</td>
</tr>
<tr>
<td>ON X GOTO 110, 120, 130</td>
<td>GOTO 100 + 10 * X</td>
</tr>
<tr>
<td>ON X GOSUB 1100, 1200, 1300</td>
<td>GOSUB 1000 + 100 * X</td>
</tr>
<tr>
<td>HOME</td>
<td>CALL -936</td>
</tr>
<tr>
<td>INVERSE</td>
<td>POKE 50, 127</td>
</tr>
<tr>
<td>NORMAL</td>
<td>POKE 50, 255</td>
</tr>
<tr>
<td>FLASH</td>
<td>POKE 50, 63</td>
</tr>
<tr>
<td>X% (integer variable)</td>
<td>X</td>
</tr>
<tr>
<td>&lt;&gt; or &gt;&lt;</td>
<td>#</td>
</tr>
</tbody>
</table>

Other Differences

As the name implies, the only numbers Integer BASIC can deal with are integers (whole numbers). Real variables and constants (numbers with decimal points or exponents) are available in Applesoft but not in Integer BASIC.

In Integer BASIC, the correctness of a statement’s syntax is checked when the statement is typed (when you press the RETURN key). In Applesoft, such checking is not done until the statement is executed.

Integer BASIC permits the line number in a GOTO or GOSUB statement to be specified by an arithmetic variable or expression; in Applesoft it may be specified only by an actual line number.

In Applesoft, only the first two characters in a variable name are significant (for example, GOOD and GOU GE are recognized as the same variable). In Integer BASIC, all characters in a variable name are significant.

String operations are defined differently in the two languages. In Integer BASIC, both strings and arrays must be defined in a DIM statement; in Applesoft, only arrays must be so defined.
Applesoft arrays may be multidimensional; Integer BASIC arrays are limited to one dimension. There are no string arrays in Integer BASIC.

Applesoft automatically sets all array elements to zero or the null string on executing RUN or CLEAR. In Integer BASIC, your program must explicitly set all array elements to their initial values.

In Integer BASIC, if the condition specified in an IF...THEN statement is false, only the THEN portion of that statement is skipped. In Applesoft, all statements following the keyword THEN on the remainder of the same program line are skipped; program execution proceeds with the next numbered line.

In Applesoft, the TRACE statement displays the line number of each individual statement executed; in Integer BASIC, the line number is displayed just once for each program line.

In Applesoft, PEEK, POKE, and CALL may use the true range of memory addresses (0 to 65535). In Integer BASIC, locations with addresses greater than 32767 must be referred to by their corresponding negative values (location 32768 is called -32767 - 1; 32769 is called -32767; 32770 is called -32766; and so on).

If control reaches the end of an Integer BASIC program without an END statement having been executed, an error message is displayed; in Applesoft, the END statement at the end of a program is optional.

In Integer BASIC, every NEXT statement must include a variable name; in Applesoft, the variable name is optional.

In the Integer BASIC INPUT statement, the string representing the optional prompting message is followed by a comma, not a semicolon as in Applesoft. If the first variable in the INPUT list is an arithmetic (integer) variable, a question mark (?) is displayed whether the optional prompting string is present or not; if the first variable in the list is a string variable, no question mark is displayed, again whether the prompting string is present or not. In Applesoft, the question mark is displayed only if no prompting string is specified.
Converting BASIC Programs to Applesoft

Although different versions of BASIC are generally similar, there are some incompatibilities that you should know about if you’re planning to convert programs to Applesoft from Integer or other versions of BASIC. Here are some things to watch for:

- Some versions of BASIC use square brackets [ ] to denote array subscripts; Applesoft uses parentheses ( )

- Many versions of BASIC require that you define the lengths of all strings in DIM statements before you use them. In converting a program to Applesoft, remove all such DIM statements for strings; use DIM only to define arrays. In some of these other versions of BASIC, a statement of the form

  \[
  \text{DIM A$ (I, J)}
  \]

defines a string array of J elements, each of length I. Convert DIM statements of this type to

  \[
  \text{DIM A$ (J)}
  \]

- Some versions of BASIC use a comma (,) or an ampersand (&) for string concatenation; Applesoft uses a plus sign (+).

- Applesoft uses the string functions LEFT$, MID$, and RIGHT$ to extract substrings. Other versions of BASIC (such as Integer BASIC) use the expression

  \[
  \text{A$ (I)}
  \]

  to refer to character number I of string A$, and

  \[
  \text{A$ (I, J)}
  \]

  to designate the substring of A$ from character number I to character number J. These expressions occurring on the right side of an assignment statement can be converted to Applesoft as follows:

  Convert \[
  \text{A$ (I)}
  \]
  to \[
  \text{MID$ (A$, I, 1)}
  \]

  Convert \[
  \text{A$ (I, J)}
  \]
  to \[
  \text{MID$ (A$, I, J - I + 1)}
  \]
When these expressions occur on the left side of an assignment statement, convert them as follows:

Convert \( A$(I) = X$ 

\[ \text{to } A$ = \text{LEFT$(A$, I - 1) + X$ + MID$(A$, I + 1) \]

Convert \( A$(I, J) = X$ 

\[ \text{to } A$ = \text{LEFT$(A$, I - 1) + X$ + MID$(A$, J + 1) \]

- Some versions of BASIC allow "multiple assignment" statements of the form

\[ \text{LET } B = C = 0 \]

This statement would set both variables \( B \) and \( C \) to 0.

In Applesoft, such a statement has an entirely different effect: all equal signs after the first are interpreted as logical comparison operators. Thus the statement above will set variable \( B \) to 1 (meaning "true") if \( C \) equals 0, to 0 (meaning "false") if it doesn't.

To convert such a multiple assignment statement into Applesoft, rewrite it as

\[ C = 0 : B = C \]

or

\[ B = 0 : C = 0 \]

- Some versions of BASIC use a slash (/) instead of a colon (:) to separate multiple statements on the same line. In converting to Applesoft, change each such slash to a colon.

- Programs that use the MAT (matrix arithmetic) functions available in some versions of BASIC will have to be rewritten using FOR/NEXT loops to perform the corresponding matrix operations.

Comparison with Integer BASIC
This appendix discusses Applesoft's facilities for storing programs and information on tape cassettes. For any of these features to work, all of the following conditions must be present:

- There must be a cassette tape recorder properly connected to the computer.
- The tape recorder must be turned on.
- There must be a tape cassette properly mounted in the recorder.
- The recorder must be set to "record" or "play," depending on the statement being executed.

None of the Applesoft tape operations checks for these conditions; if any of the conditions doesn't hold, the system may hang indefinitely. Only \texttt{CONTROL} \texttt{-} \texttt{RESET} can interrupt a tape operation; only God can make a tree.

\textbf{The SAVE Command}

\texttt{SAVE}

The \texttt{SAVE} command writes the Applesoft program currently in memory onto a tape cassette. No prompting message or signal of any kind is given; the tape recorder must already be turned on and set to "record" at the time the \texttt{SAVE} command is executed. Beeps signal the start and end of the recording.

Occasionally a tape recorder will not work properly when both input and output cables are plugged in at the same time. This problem originates from a ground loop in the tape recorder itself, which prevents making a good recording. The easiest solution is to unplug the output cable (usually labeled "monitor" on the tape recorder) when recording. Such a ground loop causes no trouble when reading a tape.
If your system is equipped with a disk drive and you have the Disk Operating System (DOS) loaded and running, a `SAVE` command with a name following the keyword `SAVE` will write the current program onto a disk under that file name. See Section 1.2.5 and your DOS manual for more information.

**The LOAD Command**

```plaintext
LOAD
```

The `LOAD` command reads an Applesoft program into memory from a tape cassette. No prompting message or signal of any kind is given; the tape recorder must already be turned on and set to “play” at the time the `LOAD` command is executed. A beep signals when the beginning of information is detected on the tape; a second beep is sounded when the program has been successfully loaded.

If your system is equipped with a disk drive and you have the Disk Operating System (DOS) loaded and running, a `LOAD` command with a name following the keyword `LOAD` will read a program from a disk under that file name. See Section 1.2.6 and your DOS manual for more information.

**The STORE Statement**

```plaintext
STORE
```

The `STORE` statement writes the contents of an integer or real array onto a tape cassette. The name of the array (e.g., `MX` in the example above) follows the keyword `STORE`, without a subscript. No prompting message or signal of any kind is given; the tape recorder must already be turned on and set to “record” at the time the `STORE` statement is executed. Beeps signal the start and end of the recording.

String arrays cannot be written with the `STORE` statement.

**The RECALL Statement**

```plaintext
RECALL
```

The `RECALL` statement reads information into an integer or real array from a tape cassette. The name of the array (e.g., `MX` in the example above) follows the keyword `RECALL`, without a subscript. The designated array must have been previously defined in a `DIM` statement in the program issuing the `RECALL`.

If You Have a Cassette Recorder
No prompting message or signal of any kind is given; the tape recorder must already be turned on and set to "play" at the time the RECALL statement is executed. A beep signals when the beginning of information is detected on the tape; a second beep is sounded when the information has been successfully transferred.

String arrays cannot be read with the RECALL statement.

The name of the array read with RECALL need not be the same name used in the STORE statement that wrote the information onto the tape. However, the dimensions of the array being read should be the same as those of the array originally written. For example, if the tape was written by the statement

```
STORE A
```

where array A had been defined by

```
DIM A (5, 5, 5)
```

it can be read back with the statement

```
RECALL B
```

where array B is defined by

```
DIM B (5, 5, 5)
```

If the dimensions of the two arrays differ, RECALL may scramble the information read into array B, or the program may halt with the message

```
?OUT OF MEMORY ERROR
```

**The SHLOAD Statement**

**SHLOAD**

The SHLOAD (for "shape load") statement reads a shape table into memory from a tape cassette. The shape table is loaded just below the current setting of HIMEM : (see Section 7.2.1, "The HIMEM : Statement") and HIMEM : is reset to just below the shape table to protect it.

No prompting message or signal of any kind is given; the tape recorder must already be turned on and set to "play" at the time the SHLOAD command is executed. A beep signals when the beginning of information is detected on the tape; a second beep is sounded when the shape table has been successfully loaded.

See Section 6.3 for extensive information on shape tables.
If You Have a Cassette Recorder
Complete Listing of the Postage Rates Program

Below is a complete listing of the postage rates program developed in Chapter 8. A copy of this program is included on the APPLESOFT SAMPLER disk.

10 REM POSTAGE RATES—name of program
20 : —colon leaves line empty
30 REM DETERMINES POSTAGE FEES —what program does
40 REM FOR EXPRESS, 1ST CLASS,
50 REM AND PRIORITY MAIL —empty line inserted by embedding [CONTROL] -J (line feed) at end of REM statement in line 50

60 REM V29/01/82 —number and date of this version
70 REM BY JOHN SCRIBBLEMONGER —programmer's credit line

100 REM MENU OF POSTAGE CLASSES —[CONTROL] -J here
110 HOME —begin with a clear screen
120 TITLE$ = "POSTAGE RATES"
130 PRINT
140 HTAB 21-LEN (TITLE$) / 2 —formula to center title
150 PRINT TITLE$
160 VTAB 6
170 PRINT "1. EXPRESS"
180 PRINT "2. FIRST CLASS"
190 PRINT "3. PRIORITY"
200 PRINT
210 PRINT "4. END THE PROGRAM" —the escape hatch

Complete Listing of the Postage Rates Program
300 REM
  GET CLASS OF MAIL
  —CONTROL-J here

310 VTAB 14
320 PRINT "Press the number of your choice:";
  —semicolon keeps response on same line
330 GET C$
  —only one keypress needed; cuts down on error possibilities. Note use of string variable to get number; avoids type mismatch errors
  —CONTROL-J here

335 REM
  CHECK FOR VALIDITY
  —another CONTROL-J (last time this is noted)
340 IF C$ = "4" THEN END
  —end program if user types a 4
350 IF VAL (C$) > 0 AND VAL (C$) < 4
  THEN 380
  —skip next two lines if valid choice typed
360 PRINT CHR$(7); CHR$(7);
  —beep twice to get attention
370 GOTO 330
  —response was invalid; try again
380 PRINT C$
  —since choice accepted via GET, it isn't displayed on the screen. Display it back to user
390 C = VAL (C$)
  —need this value later to determine what section of program to branch to for proper processing

500 REM
  GET WEIGHT OF ITEM
505 VTAB 16
510 PRINT "Please enter the WEIGHT - a number plus an O (for ounces) or a P (for pounds) - and press the
  RETURN key:";
  —prompting message to tell user what information to type and how to type it
520 CALL -868
  —clear to end of line; useful to erase any errors that might be typed
530 INPUT ""; W$  — semicolon suppresses question mark
540 W1$ = RIGHT$ (W$, 1)  — rightmost letter should be either 0 or P; use it later to see if weight is consistent with postal regulations
550 W = VAL (W$)  — how many ounces or pounds?
555 REM
560 IF W > 0 AND (W1$ = "0" OR W1$ = "P") THEN 710  — if a weight was typed, and if last character was either 0 for ounces or P for pounds, then proceed
570 PRINT CHR$ (7); CHR$ (7)  — beep twice to get attention
580 GOTO 500  — entry was invalid; try again
590 REM
600 CHECK CONSISTENCY
710 ON C GOSUB 10000, 11000, 12000  — branch to appropriate subroutine to see if weight typed is within postal rules or program limitations for mail class chosen
720 IF NOT EFLAG THEN 910  — if no inconsistency detected in subroutine then proceed with processing
730 GOSUB 60000 : REM KEYSTALL  — wait for user to acknowledge message
740 EFLAG = 0  — clear error flag set in subroutine
750 CLEAR  — reset all variables, clear arrays, etc.
760 GOTO 100  — restart program loop
770 REM
780 FIND APPROPRIATE CODE FOR PROCESSING  — everything is valid and consistent; now program can solve for the postage rate!
910 ON C GOSUB 1000, 2000, 3000
   —branch to proper calculating routine
920 GOSUB 61000 : REM FORMATTER
   —format result for display
930 PRINT
935 REM
   DISPLAY RESULTS
940 PRINT "POSTAGE NEEDED: $"; T$
   —finally, the postage due!
950 GOSUB 60000 : REM KEYSTALL
   —don’t go on until user is ready
960 CLEAR
970 GOTO 100
   —...and do it
999 REM
   SUBROUTINES BEGIN HERE
1000 REM
   EXPRESS MAIL CALCULATION
1010 W = INT (W + .88)
   —weight must be increased to compensate for fractions;
   —postal rates read “NOT MORE THAN x POUNDS”
1020 T = R (W)
   —rate array filled in express mail consistency-checking routine
   —(line 10000)
1030 RETURN
   —end routine
2000 REM
   FIRST CLASS CALCULATION
2010 T = .20 + INT (W + .99 - 1) * .17
   —first class rate is 20 cents first ounce plus 17 cents for each additional ounce or portion thereof (April, 1982 rates)
2020 RETURN
   —end routine
3000 REM
   PRIORITY MAIL CALCULATION
3010 W = INT (W + .99)
   —compensate for partial ounces or pounds
3020 IF W > 10 THEN 3160
—go to line 3160 for weights greater than 10 pounds (ounce weights converted to pounds in consistency subroutine starting at line 12000)

3025 REM
PRIORITY RATES TO 10 POUNDS
3030 IF W <= 1 THEN T = 2.24
3040 IF W > 1 AND W <= 1.5 THEN T = 2.30
—rates in half-pound increments
3050 IF W > 1.5 AND W <= 2 THEN T = 2.54
3060 IF W > 2 AND W <= 2.5 THEN T = 2.78
3070 IF W > 2.5 AND W <= 3 THEN T = 3.01
3072 IF W > 3 AND W <= 3.5 THEN T = 3.25
3078 IF W > 3.5 AND W <= 4 THEN T = 3.49
3080 IF W > 4 AND W <= 4.5 THEN T = 3.73
3090 IF W > 4.5 AND W <= 5 THEN T = 3.97
3100 IF W > 5 AND W <= 6 THEN T = 4.44
—rates by the pound now!
3110 IF W > 6 AND W <= 7 THEN T = 4.92
3120 IF W > 7 AND W <= 8 THEN T = 5.39
3130 IF W > 8 AND W <= 9 THEN T = 5.87
3140 IF W > 9 THEN T = 6.35
3150 GOTO 3240     —branch to RETURN statement
3160 REM
PRIORITY RATES FOR OVER 10 POUNDS
3170 T1 = INT (W / 5 - 1) * 2.38 + 3.97
—first 5 pounds cost $3.97; each added 5 pounds cost $2.38
3180 W1 = W - INT (W / 5) * 5
—how many odd pounds are there (pounds that are not multiples of 5 and must be charged at a special rate)?
3190 IF W1 = 1 THEN T2 = .47
3200 IF W1 = 2 THEN T2 = .95

Complete Listing of the Postage Rates Program
3210 IF W1 = 3 THEN T2 = 1.42
3220 IF W1 = 4 THEN T2 = 1.90
3230 T = T1 + T2  — add the 5-pound-multiples rate to the odd-pounds rate
3240 RETURN  — end routine
10000 REM
EXPRESS MAIL CONSISTENCY CHECK
10010 DATA 9.35, 9.35, 9.55, 9.90, 10.30, 10.65, 11.00, 11.40, 11.75, 0  — express mail rates; 0 at end is “last item” flag
10020 X = 0  — set up counter to check how many rates are read from DATA list
10030 X = X + 1  — increment counter
10040 READ R(X)  — put price into proper array element
10050 IF R(X) = 0 THEN 10070  — price of 0 marks end of list
10060 GOTO 10030  — get next price
10070 X = X - 1  — X includes count of “last item” flag from 10050; subtract it from count since it’s a “dummy” item
10080 IF W1$ = "P" THEN 10100  — next line is for ounces only
10090 W = W / 16  — convert ounces to pounds
10100 IF W <= X THEN 10140  — if weight in pounds is covered by the rate chart, then go ahead
10110 PRINT
10120 PRINT CHR$(7); CHR$(7); "TOO HEAVY FOR MY TABLES - PLEASE CALL THE POST OFFICE"
10130 EFLAG = 1  — set flag indicating inconsistent weight/type; will be checked at line 720
10140 RETURN  — end routine
11000 REM
FIRST CLASS CONSISTENCY CHECK
11010 IF W1$ = "O" AND W < 12.01 THEN
11060  — OK if not more than 12 ounces
11020 PRINT
11030 PRINT CHR$ (7); CHR$ (7); "TOO HEAVY FOR FIRST CLASS"
          —sorry—inconsistent!
11040 PRINT "TRY PRIORITY MAIL"
          —suggest alternative
11050 EFLAG = 1
          —set flag indicating inconsistent weight/type; will be checked at line 720
11060 RETURN
          —end routine

12000 REM
PRIORITY MAIL CONSISTENCY CHECK
12010 IF W1$ = "P" THEN 12090
          —if in pounds, then skip down
12020 IF W > 12 THEN 12080
          —skip down if weight is between 12 and 16 ounces
12030 PRINT
12040 PRINT CHR$ (7); CHR$ (7); "TOO LIGHT FOR PRIORITY MAIL -"
          —too light!
12050 PRINT "TRY FIRST CLASS"
          —suggest alternative
12060 EFLAG = 1
          —set flag indicating inconsistent weight/type; will be checked at line 720
12070 GOTO 12150
          —branch to end of routine
12080 W = W / 16
          —convert ounces to pounds
12090 IF W <= 70 THEN 12150
          —final check: is item on the charts?
12100 PRINT
12110 PRINT CHR$ (7); CHR$ (7); "TOO HEAVY FOR PRIORITY MAIL -"
          —off the charts
12120 PRINT "TRY ONE OF THE AIR EXPRESS COMPANIES"
          —too big for the Post Office!
12130 EFLAG = 1
          —set flag indicating inconsistent weight/type; will be checked at line 720
12150 RETURN
          —end routine
UTILITY ROUTINES — routines useful for various tasks but ancillary to rest of program

KEYSTALL — routine to interrupt program until user presses a key

VTAB 24 — move cursor to screen bottom

INVERSE — set text to appear black-on-white

PRINT "PRESS RETURN TO GO ON..." ; — wait for keypress

GET A$ — restore ordinary white-on-black

RETURN — end routine

MONEY FORMATTER — adds zeros after the decimal point where needed

T$ = STR$ (T) — turn the calculated postage fee into a string

IF T = INT (T) THEN T$ = T$ + ".00" — if charge is in whole dollars, add a decimal point and two zeros

IF ASC (RIGHT$ (T$, 2)) = 46 THEN T$ = T$ + "0" — if second character from the right is a decimal point (ASCII code 46) then number has only one digit to right of decimal—so add a "0" to the string

RETURN — end the routine
address: A number used to identify something, such as a location in the computer's memory.

algorithm: A step-by-step procedure for solving a problem or accomplishing a task.

AND: A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare OR, NOT.

Apple Ile: A personal computer in the Apple II family, manufactured and sold by Apple Computer.

Apple Ile 80-Column Text Card: A peripheral card made and sold by Apple Computer that plugs into the Apple Ile's auxiliary slot and converts the computer's display of text from 40- to 80-column width.

Apple Ile Extended 80-Column Text Card: A peripheral card made and sold by Apple Computer that plugs into the Apple Ile's auxiliary slot and converts the computer's display of text from 40- to 80-column width while extending its memory capacity by 64K bytes.

Applesoft: An extended version of the BASIC programming language used with the Apple Ile computer and capable of processing numbers in floating-point form. An interpreter for creating and executing programs in Applesoft is built into the Apple Ile system in firmware. Compare Integer BASIC.

application program: A program that puts the resources and capabilities of the computer to use for some specific purpose or task, such as word processing, data-base management, graphics, or telecommunications. Compare system program.
**application software**: The component of a computer system consisting of application programs.

**argument**: The value on which a function operates.

**arithmetic operator**: An operator, such as +, that combines numeric values to produce a numeric result; compare **relational operator**, **logical operator**.

**array**: A collection of variables referred to by the same name and distinguished by means of numerical subscripts.

**ASCII**: American Standard Code for Information Interchange; a code in which the numbers from 0 to 127 stand for text characters, used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

**assembler**: A language translator that converts a program written in assembly language into an equivalent program in machine language.

**assembly language**: A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

**auxiliary slot**: The special expansion slot inside the Apple Ile used for the Apple 80-Column Text Card or Extended 80-Column Text Card.

**back panel**: The rear face of the Apple Ile computer, which includes the power switch, the power connector, and connectors for a video display device, a cassette tape recorder, and other peripheral devices.

**BASIC**: Beginner’s All-purpose Symbolic Instruction Code; a high-level programming language designed to be easy to learn and use. Two versions of BASIC are available from Apple Computer for use with the Apple Ile: Applesoft (built into the Apple Ile in firmware) and Integer BASIC (provided on the DOS 3.3 SYSTEM MASTER disk).

**binary**: The representation of numbers in terms of powers of two, using the two digits 0 and 1. Commonly used in computers, since the values 0 and 1 can easily be represented in physical form in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen.
binary file: A file containing "raw" information not expressed in text form; compare text file.

binary operator: An operator that combines two operands to produce a result; for example, + is a binary arithmetic operator, < is a binary relational operator, and 0R is a binary logical operator. Compare unary operator.

bit: A binary digit (0 or 1); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

bit bucket: The final resting place of all information; see write-only memory.

body: The statements or instructions making up some construct in a program, such as a loop or a subroutine.

boot: To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to "pull itself in by its own bootstraps"; hence the term bootstrapping or booting.

boot disk: See startup disk.

bootstrap: See boot.

branch: To send program execution to a line or statement other than the next in sequence.

buffer: An area of the computer's memory reserved for a specific purpose, such as to hold graphical information to be displayed on the screen or text characters being read from some peripheral device. Often used as an intermediary "holding area" for transferring information between devices operating at different speeds, such as the computer's processor and a printer or disk drive. Information can be stored into the buffer by one device and then read out by the other at a different speed.

bug: An error in a program that causes it not to work as intended.

byte: A unit of information consisting of a fixed number of bits; on the Apple IIe, one byte consists of eight bits and can hold any value from 0 to 255.
call: To request the execution of a subroutine or function.

card: See peripheral card.

catalog: A list of all files stored on a disk; sometimes called a directory.

cathode-ray tube: An electronic device, such as a television picture tube, that produces images on a screen coated with phosphors that emit light when struck by a focused beam of electrons.

central processing unit: See processor.

color: A letter, digit, punctuation mark, or other written symbol used in printing or displaying information in a form readable by humans.

character code: A number used to represent a text character for processing by a computer system.

code: (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

command: A communication from the user to a computer system (usually typed from the keyboard) directing it to perform some immediate action.

compiler: A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) for later execution. Compare interpreter.

component: A part; in particular, a part of a computer system.

computer: An electronic device for performing predefined (programmed) computations at high speed and with great accuracy.

computer system: A computer and its associated hardware, firmware, and software.

concatenate: Literally, "to chain together"; to combine two or more strings into a single, longer string containing all the characters in the original strings.
**conditional branch:** A branch that depends on the truth of a condition or the value of an expression; compare unconditional branch.

**constant:** A symbol in a program representing a fixed, unchanging value; compare variable.

**control:** The order in which the statements of a program are executed.

**control character:** A character that controls or modifies the way information is printed, transmitted or displayed. Control characters have ASCII codes between 0 and 31 and are typed from the Apple Ile keyboard by holding down the [CONTROL] key while typing some other character. For example, the character [CONTROL]-C (ASCII code 3) means "interrupt program execution."

**controller card:** A peripheral card that connects a device such as a printer or disk drive to the Apple Ile and controls the operation of the device.

**control variable:** See index variable.

**CPU:** Central processing unit; see processor.

**crash:** To cease operating unexpectedly, possibly damaging or destroying information in the process.

**CRT:** See cathode-ray tube.

**current input device:** The source, such as the keyboard or a modem, from which an Applesoft program is currently receiving its input.

**current output device:** The destination, such as the display screen or a printer, to which an Applesoft program is currently sending its output.

**cursor:** A marker or symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

**data:** Information; especially information used or operated on by a program.

**debug:** To locate and correct an error or the cause of a problem or malfunction in a computer program.
decimal: The common form of number representation used in everyday life, in which numbers are expressed in terms of powers of ten, using the ten digits 0 to 9.

default: (1) A value, action, or setting that is automatically used by a computer system when no other explicit information has been given. For example, if a command to run a program from a disk does not identify which disk drive to use, the Disk Operating System will automatically use the same drive that was used in the last operation. (2) That which, dear Brutus, is not in our stars.

defered execution: The saving of an Applesoft program line for execution at a later time as part of a complete program; occurs when the line is typed with a line number. Compare immediate execution.

delimiter: A character that is used for punctuation to mark the beginning or end of a sequence of characters, and which therefore is not considered part of the sequence itself. For example, Applesoft uses the double quotation mark (" ) as a delimiter for string constants: the string "DOG" consists of the three characters D, O, and G, and does not include the quotation marks. In written English, the space character is used as a delimiter between words.

device: (1) A physical apparatus for performing a particular task or achieving a particular purpose. (2) In particular, a hardware component of a computer system.

digit: (1) One of the characters 0 to 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 to 9 and A to F in hexadecimal.

dimension: The maximum size of one of the subscripts of an array.

directory: A list of all files stored on a disk; sometimes called a catalog.

disk: An information storage medium consisting of a flat, circular magnetic surface on which information can be recorded in the form of small magnetized spots, similarly to the way sounds are recorded on tape.

disk drive: A peripheral device that writes and reads information on the surface of a magnetic disk.
diskette: A term sometimes used for the small (5-1/4-inch) flexible disks used with the Apple Disk II drive.

Disk II drive: A model of disk drive made and sold by Apple Computer for use with the Apple IIe computer; uses 5-1/4-inch flexible ("floppy") disks.

Disk Operating System: An optional software system for the Apple IIe that enables the computer to control and communicate with one or more Disk II drives.

disk-resident: Stored or held permanently on a disk.

display: (1) Information exhibited visually, especially on the screen of a display device. (2) To exhibit information visually. (3) A display device.

display color: The color currently being used to draw high- or low-resolution graphics on the display screen.

display device: A device that exhibits information visually, such as a television receiver or video monitor.

display screen: The glass or plastic panel on the front of a display device, on which images are displayed.

DOS: See Disk Operating System.

edit: To change or modify; for example, to insert, remove, replace, or move text in a document.

element: A member of a set or collection; specifically, one of the individual variables making up an array.

embedded: Contained within. For example, the string "HUMPTY DUMPTY" is said to contain an embedded space.

ending value: The value against which the index variable is tested after each pass through a loop, to determine when to stop repeating the loop.

error code: A number or other symbol representing a type of error.

error message: A message displayed or printed to notify the user of an error or problem in the execution of a program.
escape mode: A state of the Apple IIe computer, entered by pressing the \[ESC\] key, in which certain keys on the keyboard take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes beginning with the \[ESC\] key, used for positioning the cursor and controlling the display of text on the screen.

execute: To perform or carry out a specified action or sequence of actions, such as those described by a program.

expansion slot: A connector inside the Apple IIe computer in which a peripheral card can be installed; sometimes called peripheral slot.

expression: A formula in a program describing a calculation to be performed.

FIFO: First in, first out.

file: A collection of information stored as a named unit on a peripheral storage medium such as a disk.

file name: The name under which a file is stored.

firmware: Those components of a computer system consisting of programs stored permanently in read-only memory. Such programs (for example, the Applesoft interpreter and the Apple IIe Monitor program) are built into the computer at the factory; they can be executed at any time but cannot be modified or erased from main memory. Compare hardware, software.

fixed-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number, so that the number is interpreted as an integer. Fixed-point numbers of a given length cover a narrower range than floating-point numbers of the same length, but with greater precision. Compare floating-point.

flag: A variable whose contents (usually 1 or 0, standing for true or false) indicate whether some condition holds or whether some event has occurred, used to control the program's actions at some later time.
**floating-point**: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to "float" to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Floating-point numbers of a given length cover a wider range than fixed-point numbers of the same length, but with less precision. Compare **fixed-point**.

**format**: (1) The form in which information is organized or presented. (2) To specify or control the format of information. (3) To prepare a blank disk to receive information by dividing its surface into tracks and sectors; also *initialize*.

**function**: A preprogrammed calculation that can be carried out on request from any point in a program.

**GAME I/O connector**: A special 16-pin connector inside the Apple IIe, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector**.

**graphics**: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

**hand control**: An optional peripheral device that can be connected to the Apple IIe's hand control connector and has a rotating dial and a pushbutton; typically used to control game-playing programs, but can be used in more serious applications as well.

**hand control connector**: A 9-pin connector on the Apple IIe's back panel, used for connecting hand controls to the computer. Compare **GAME I/O connector**.

**hang**: For a program or system to "spin its wheels" indefinitely, performing no useful work.

**hard copy**: Information printed on paper for human use.

**hardware**: Those components of a computer system consisting of physical (electronic or mechanical) devices. Compare **software**, **firmware**.
hertz: The unit of frequency of vibration or oscillation, also called cycles per second; named for the physicist Heinrich Hertz and abbreviated Hz. The current provided by a standard power outlet alternates at a rate of 60 hertz; that is, it changes polarity 60 times each second. The Apple Ile’s 6502 microprocessor operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

hexadecimal: The representation of numbers in terms of powers of sixteen, using the sixteen digits 0 to 9 and A to F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form: each hexadecimal digit corresponds to a sequence of four binary digits, or bits.

high-level language: A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language.

high-order byte: The more significant half of a memory address or other two-byte quantity. In the Apple Ile’s 6502 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

high-resolution graphics: The display of graphics on the Apple Ile’s display screen as a six-color array of points, 280 columns wide and 192 rows high.

Hz: See hertz.

immediate execution: The execution of an Applesoft program line as soon as it is typed; occurs when the line is typed without a line number. Compare deferred execution.

implement: To realize or bring about; for example, a language translator implements a particular language.

infinite loop: A section of a program that will repeat the same sequence of actions indefinitely.

information: Facts, concepts, or instructions represented in an organized form.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position.
index variable: A variable whose value changes on each pass through a loop; often called control variable or loop variable.

initialize: (1) To set to an initial state or value in preparation for some computation. (2) To prepare a blank disk to receive information by dividing its surface into tracks and sectors; also format.

input: (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer: A whole number, with no fractional part; represented inside the computer in fixed-point form. Compare real number.

Integer BASIC: A version of the BASIC programming language used with the Apple II family of computers; older than Applesoft and capable of processing numbers in integer (fixed-point) form only. An interpreter for creating and executing programs in Integer BASIC is included on the DOS 3.3 SYSTEM MASTER disk, and is automatically loaded into the computer's memory when the computer is started up with that disk. Compare Applesoft.

interactive: Operating by means of a dialog between the computer system and a human user.

interface: The devices, rules, or conventions by which one component of a system communicates with another.

interpreter: A language translator that reads a program written in a particular programming language and immediately carries out the actions that the program describes. Compare compiler.

inverse video: The display of text on the computer's display screen in the form of black dots on a white (or other single phosphor color) background, instead of the usual white dots on a black background.

I/O: Input/output; the transfer of information into and out of a computer. See input, output.

I/O device: Input/output device; a device that transfers information into or out of a computer. See input, output, peripheral device.
K: Two to the tenth power, or 1024 (from the Greek root kilo, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

keyboard: The set of keys built into the Apple IIe computer, similar to a typewriter keyboard, for typing information to the computer.

keystroke: The act of pressing a single key or a combination of keys (such as CONTROL-C) on the Apple IIe keyboard.

keyword: A special word or sequence of characters that identifies a particular type of statement or command, such as RUN or PRINT.

kilobyte: A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see K.

language: See programming language.

language translator: A system program that reads a program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See interpreter, compiler, assembler.

LIFO: Last in, first out.

line: See program line.

line number: A number identifying a program line in an Applesoft program.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use; for example, to transfer a program into memory for execution.

location: See memory location.

logical operator: An operator, such as AND, that combines logical values to produce a logical result; compare arithmetic operator, relational operator.

loop: A section of a program that is executed repeatedly, usually until some condition is met (such as an index variable reaching a specified ending value).

loop variable: See index variable.
**low-level language:** A programming language that is relatively close to the form that the computer's processor can execute directly.

**low-order byte:** The less significant half of a memory address or other two-byte quantity. In the Apple Ile's 6502 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

**low-resolution graphics:** The display of graphics on the Apple Ile's display screen as a sixteen-color array of blocks, 40 columns wide and 48 rows high.

**machine language:** The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 6502 microprocessor used in the Apple Ile) has its own form of machine language.

**main memory:** The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

**mask:** A pattern of bits for use in bit-level logical operations.

**memory:** A hardware component of a computer system that can store information for later retrieval; see main memory, random-access memory, read-only memory, read-write memory, write-only memory.

**memory location:** A unit of main memory that is identified by an address and can hold a single item of information of a fixed size; in the Apple Ile, a memory location holds one byte, or eight bits, of information.

**memory-resident:** (1) Stored permanently in main memory, as firmware. (2) Held continually in main memory even while not in use, as the Disk Operating System.

**menu:** A list of choices presented by a program, usually on the display screen, from which the user can select.

**MHz:** Megahertz; one million hertz. See hertz.

**microcomputer:** A computer, such as the Apple Ile, whose processor is a microprocessor.
**microprocessor**: A computer processor contained in a single integrated circuit, such as the 6502 microprocessor used in the Apple IIe.

**mode**: A state of a computer or system that determines its behavior.

**modem**: Modulator/demodulator; a peripheral device that enables the computer to transmit and receive information over a telephone line.

**monitor**: See *video monitor*.

**Monitor program**: A system program built into the Apple IIe in firmware, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

**nested loop**: A loop contained within the body of another loop and executed repeatedly during each pass through the containing loop.

**nested subroutine call**: A call to a subroutine from within the body of another subroutine.

**nibble**: A unit of information equal to half a byte, four bits, or fifty cents; can hold any value from 0 to 15. Sometimes spelled *nybble*.

**NOT**: A unary logical operator that produces a true result if its operand is false, a false result if its operand is true; compare AND, OR.

**null string**: A string containing no characters.

**operand**: A value to which an operator is applied.

**operating system**: A software system that organizes the computer's resources and capabilities and makes them available to the user or to application programs running on the computer.

**operator**: A symbol or sequence of characters, such as + or AND, specifying an operation to be performed on one or more values (the operands) to produce a result; see arithmetic operator, relational operator, logical operator, unary operator, binary operator.

**OR**: A logical operator that produces a true result if either or both of its operands are true, a false result if both of its operands are false; compare AND, NOT.
**output**: (1) Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem. (2) The act or process of transferring such information.

**page**: (1) A screenful of information on a video display, consisting on the Apple Ile of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256 bytes.

**pass**: A single execution of a loop.

**peek**: To read information directly from a location in the computer’s memory.

**peripheral**: At or outside the boundaries of the computer itself, either physically (as a peripheral device) or in a logical sense (as a peripheral card).

**peripheral card**: A removable printed-circuit board that plugs into one of the Apple Ile’s expansion slots and expands or modifies the computer’s capabilities by connecting a peripheral device or performing some subsidiary or peripheral function.

**peripheral device**: A device, such as a video monitor, disk drive, printer, or modem, used in conjunction with a computer. Often (but not necessarily) physically separate from the computer and connected to it by wires, cables, or some other form of interface, typically by means of a peripheral card.

**peripheral slot**: See expansion slot.

**plotting vector**: A code in a shape definition representing a single step in drawing a shape on the high-resolution graphics screen, specifying whether to plot a point at the current screen position and in what direction to move (up, down, left, or right) before processing the next vector. See shape definition, shape table.

**point of call**: The point in a program from which a subroutine or function is called.

**pointer**: An item of information consisting of the memory address of some other item. For example, Applesoft maintains internal pointers to (among other things) the most recently stored variable, the most recently typed program line, and the most recently read DATA item.
**poke**: To store information directly into a location in the computer's memory.

**pop**: To remove the top entry from a stack.

**precedence**: The order in which operators are applied in evaluating an expression.

**printed-circuit board**: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, to which integrated circuits and other electronic components are connected.

**printer**: A peripheral device that writes information on paper in a form easily readable by humans or literate androids.

**processor**: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

**program**: (1) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. In Applesoft, a sequence of program lines, each with a different line number. (2) To write a program.

**program line**: The basic unit of an Applesoft program, consisting of one or more statements separated by colons ( : ).

**programmer**: The human author of a program; one who writes programs.

**programming**: The activity of writing programs.

**programming language**: A set of rules or conventions for writing programs.

**prompt**: To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.
prompt character: (1) A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character ] is used by the Applesoft BASIC interpreter, > by Integer BASIC, and * by the system Monitor program. Also called prompting character. (2) Someone who is always on time.

prompt message: A message displayed on the screen to prompt the user for some action. Also called prompting message.

push: To add an entry to the top of a stack.

queue: A list in which entries are added at one end and removed at the other, causing entries to be removed in FIFO (first-in-first-out) order; compare stack.

RAM: See random-access memory.

random-access memory: Memory in which the contents of individual locations can be referred to in an arbitrary or random order. This term is often used incorrectly to refer exclusively to read-write memory; but strictly speaking both read-only and read-write memory can be accessed in random order. This misuse of the term random-access is an attempt to confuse new users, creating a rite of passage and an excellent market for glossaries of computer terms. Compare read-only memory, read-write memory, write-only memory.

read: To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory: Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and can never be erased or changed. Compare read-write memory, random-access memory, write-only memory.

read-write memory: Memory whose contents can be both read and written; often misleadingly called random-access memory, or RAM. The information contained in read-write memory is erased when the computer's power is turned off, and is permanently lost unless it has been saved on a more permanent storage medium, such as a disk. Compare read-only memory, random-access memory, write-only memory.
real number: A number that may include a fractional part; represented inside the computer in floating-point form. Compare integer.

relational operator: An operator, such as >, that compares numeric values to produce a logical result; compare arithmetic operator, logical operator.

reserved word: A word or sequence of characters reserved by a programming language for some special use, and therefore unavailable as a variable name in a program.

resident: See memory-resident, disk-resident.

return address: The point in a program to which control returns on completion of a subroutine or function.

ROM: See read-only memory.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

run: (1) To execute a program. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To transfer information from main memory to a peripheral storage medium for later use.

scientific notation: A method of expressing numbers in terms of powers of ten, useful for expressing numbers that may vary over a wide range, from very small to very large. For example, the number of atoms in a gram of hydrogen is approximately \(6.02 \times 10^{23}\), meaning 6.02 times ten to the 23rd power. (The letter E stands for “exponent.”) The number is easier to understand in this form than in the form 6020000000000000000000000.

screen: See display screen.

scroll: To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See viewport, window.

seed: A value used to begin a repeatable sequence of random numbers.
**shape definition:** A coded description of a shape to be drawn on the high-resolution graphics screen, consisting of one or more plotting vectors. See **shape table, plotting vector**.

**shape table:** A collection of one or more shape definitions, together with their indices.

**shape table index:** A list giving the memory addresses of the shapes in a shape table.

**simple variable:** A variable that is not an element of an array.

**soft switch:** A means of changing some feature of the Apple Ile from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

**software:** Those components of a computer system consisting of programs that determine or control the behavior of the computer. Compare **hardware, firmware**.

**space character:** A text character whose printed representation is a blank space, typed from the keyboard by pressing the SPACE bar.

**stack:** A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in-first-out) order; compare **queue**.

**starting value:** The value assigned to the index variable on the first pass through a loop.

**startup disk:** A disk containing software recorded in the proper form to be loaded into the Apple Ile's memory in order to set the system into operation. Sometimes called a boot disk; see **boot**.

**statement:** A unit of a program in a high-level language specifying an action for the computer to perform, typically corresponding to several instructions of machine language.

**step value:** The amount by which the index variable changes on each pass through a loop.

**stepwise refinement:** A technique of program development in which broad sections of the program are laid out first, then elaborated step by step until a complete program is obtained.

**string:** An item of information consisting of a sequence of text characters.
strobe: (1) An event, such as a change in a signal, that triggers some action. (2) A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

subscript: An index number used to identify a particular element of an array.

substring: A string that is part of another string.

syntax: The rules governing the structure of statements or instructions in a programming language.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose.

system program: A program that makes the resources and capabilities of the computer available for general purposes, such as an operating system or a language translator. Compare application program.

text: (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple Ile’s display screen. Compare graphics.

text file: A file containing information expressed in text form; compare binary file.

text window: An area on the Apple Ile’s display screen within which text is displayed and scrolled.

truncate: To shorten by discarding a part; specifically, to convert a real number to the next lower integer.

unary operator: An operator that applies to a single operand; for example, the minus sign (−) in a negative number such as −6 is a unary arithmetic operator. Compare binary operator.

unconditional branch: A branch that does not depend on the truth of any condition; compare conditional branch.

user: The person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.
value: An item of information that can be stored in a variable, such as a number or a string.

variable: (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location; compare constant.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form. (3) An early space pioneer.

video monitor: A display device capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Most video monitors can be connected directly to the Apple Ile computer as a display device.

viewport: All or part of the display screen, used by an application program to display a portion of the information (such as a document, picture, or worksheet) that the program is working on. Compare window.

window: (1) The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen; compare viewport. (2) A viewport. (3) A flat, rectangular panel, usually made of silica, used in many archaic structures as a human-to-nature interface.

wraparound: The automatic continuation of text from the end of one line to the beginning of the next, as on the display screen or a printer.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

write-only memory: A form of computer memory into which information can be stored but never, ever retrieved, developed under government contract in 1975 by Professor Homberg T. Farnsfarfe. Farnsfarfe's original prototype, approximately one inch on each side, has so far been used to store more than 100 trillion words of surplus federal information. Farnsfarfe's critics have denounced his project as a six-million-dollar boondoggle, but his defenders point out that this excess information would have cost more than 250 billion dollars to store in conventional media. Compare read-only memory, read-write memory, random-access memory.
Index

A
ABS function 38, 215
absolute value 38, 215
addition 32, 36, 86
American National Standards
Institute (ANSI) 3
American Standard Code for
Information Interchange, see
ASCII
ampersand character (&) 246
AND 35, 175
animation 150
annunciators 131, 262, 263
ANSI: see American National
Standards Institute
Apple Ile 80-Column Text Card, see
80-Column Text Card
arc tangent 41, 216
argument of functions 37, 38, 125,
173, 179
argument variable 44
arithmetic functions 38
arithmetic operators 31
array(s) 26, 29, 77ff, 217, 228, 248,
249, 268, 275ff, 293ff, 298
dimensions 79, 80
elements 29, 77, 269
names 29, 77
storage 179
variables 275ff
arrow keys 18, 20
ASC function 215
ASCII (American Standard Code for
Information Interchange) 19, 82,
215, 241ff, 258
assignment statement 30, 215,
224, 251, 296
asterisk (*) 32
ATN function 41, 216
auto-repeat 19, 20

B
backslash character (\) 4, 18
BAD SUBSCRIPT error 79, 248
bell character ([CONTROL]-G) 130
LOAD command 158
body of loop 55
booting 96, 112
branch 49ff, 220
conditional 51
unconditional 50, 220
built-in arithmetic functions 38ff
CALL statement 71, 136, 216, 249,
253ff, 281, 294
CAN'T CONTINUE error 248
CAPS LOCK key 4
caret (^) 31
cassette input 110
cassette output 131, 264
Celsius 44
character codes 82
CHR$ function 91, 216
CLEAR command 9, 30, 129, 216,
294
colon (:) 5, 98ff, 105, 106, 177, 192,
246, 267, 296, 301
color, see display color
COLOR= statement 137, 216
comma (,) 98ff, 105, 113, 114, 115
commands, see names of
commands
concatenation 83, 84, 100, 251,
295
conditional branch 51
constants 268
CONT command 16, 17, 73, 216,
247, 248
color, see display color
COLOR= statement 137, 216
comma (,) 98ff, 105, 113, 114, 115
commands, see names of
commands
concatenation 83, 84, 100, 251,
295
conditional branch 51
constants 268
CONT command 16, 17, 73, 216,
247, 248
color, see display color
COLOR= statement 137, 216
comma (,) 98ff, 105, 113, 114, 115
commands, see names of
commands
concatenation 83, 84, 100, 251,
295
conditional branch 51
constants 268
CONT command 16, 17, 73, 216,
247, 248

C
CALL statement 71, 136, 216, 249,
253ff, 281, 294
CAN'T CONTINUE error 248
CAPS LOCK key 4
caret (^) 31
cassette input 110
cassette output 131, 264
Celsius 44
character codes 82
CHR$ function 91, 216
CLEAR command 9, 30, 129, 216,
294
colon (:) 5, 98ff, 105, 106, 177, 192,
246, 267, 296, 301
color, see display color
COLOR= statement 137, 216
comma (,) 98ff, 105, 113, 114, 115
commands, see names of
commands
concatenation 83, 84, 100, 251,
295
conditional branch 51
constants 268
CONT command 16, 17, 73, 216,
247, 248

CONTROL key 15, 16, 18, 241
-@ 98, 107
- B 176, 177, 181
-C 15f, 50, 58, 69, 72, 98, 107, 159, 180, 216
-G 130
-H 100, 107
-J (line feed character) 192, 193, 216, 301
-M 100, 107
-RESET 13-17, 96, 112, 161, 162, 166, 171
-S 15
-X 18, 100, 107

control
stack 10, 62ff, 71, 227, 265
statements 49f
COS function 40, 217
cosine 40, 217
crossed loops 60
current input device 104, 223
current output device 10, 113, 224, 228
cursor 4, 18ff, 97, 113, 115, 119ff, 220ff, 232, 234, 253, 254
cursor control 287-288

D
DATA statement 103, 105, 108, 217, 228, 229, 250
debugging 11, 180
DEF FN statement 44, 177, 217, 249
defered execution 4, 5, 9, 247
degrees 44
DEL command 6, 7, 217
DELETE key 7
DIM statement 79, 217, 251, 293, 295, 298
disk 12ff, 112, 156, 230
Disk Operating System (DOS) 12, 14, 16, 105, 157, 176, 265, 298
display color 137ff, 160, 216, 220ff, 231
display screen 111
division 32
DIVISION BY ZERO error 248
dollar sign ($) 26, 29, 82, 88, 251, 259
DOS (see Disk Operating System)
double quotation marks (" ) 28, 81, 99, 102, 270
down-arrow key 18, 19, 241
draw statement 151, 155, 156, 160, 161, 162, 163, 164, 218, 230, 231

E
e 42
editing 287-288
Eighty-Column Text Card 4, 112, 114, 115, 119, 124, 125, 127, 222, 254, 287ff
END statement 17, 73, 216, 218, 251, 269, 294
equal sign (=) 30, 34, 44, 129, 137, 145, 163, 246
equal to (=) 34
error
codes 68, 69, 247ff
messages 247ff
error handling routines 67ff, 229, 247, 264
restoring normal 71
escape mode 19, 287
ESC key 20, 242
-@ 20, 255
-A 20
-B 20
-C 20
-D 20
-E 20
-F 20, 255
-I 19, 20
-J 19, 20
-K 19, 20
-M 19, 20
exclusive-or 175
execution of program 16
EXP function 42, 218
expansion slot 96, 111
exponential 42, 218
exponentiation 32
expressions 31ff
EXTRA IGNORED message 99, 105

F
Fahrenheit 44
false 33ff
FILE NOT FOUND error 14
FLASH statement 127, 128, 218, 226
floating-point accumulator 173
FN keyword 45, 219
FOR statement 55ff, 219, 225, 271
FORMULA TOO COMPLEX error 248
FP command 291
fractions 33
FRE function 178, 220
free space 275
full-screen graphics 136, 138, 143, 144, 146, 221, 260
function names 44
functions 37ff, 173, 177, 229
argument of 37, 38, 125, 173, 179
built-in arithmetic 38
call 37, 38, 45
names 44
user-defined 44-45, 217
ABS 38, 215
ASC 215
ATN 41, 216
CHR$ 216
COS 40, 217
EXP 42, 218
FRE 178, 219
INT 39, 223
LEFT$ 100, 223, 249
LEN 224
LET 215
LOG 42, 224, 249
MID$ 100, 225, 249
peek 130, 131, 177, 178, 180, 247, 249, 253ff
PDL 109, 227
POS 125, 228
RIGHT$ 100, 229, 249
RND 43, 229
SCRN 141, 231
SGN 39, 231
SIN 40, 231
SPC 113, 120-121, 231, 249
SQR 40, 232, 249
STR$ 232
TAB 113, 120, 121, 123, 126, 161, 232, 233, 249, 254
TAN 4, 233
USR 172, 233
VAL 102, 105, 233

H
hand control 109, 262
hand control connector 109, 131, 262, 263
HCOLOR= statement 145, 160, 220
HGR statement 143, 145, 149, 161, 162, 220, 258, 259
HGR2 statement 144, 145, 149, 161, 162, 221, 259
high-resolution graphics 136, 140ff, 150, 176ff, 218, 220ff, 230, 261
HIMEM: statement 149, 156, 165, 176, 179, 221, 250, 275, 299
HLIN statement 139, 221
HOME statement 221, 254
HPLOT statement 146, 161, 218, 222, 262
HTAB statement 120, 122, 126, 181, 222, 254, 256
Humpty Dumpty 19

I
IF . . . THEN statement 33, 36, 52, 222, 248, 251, 267, 294
ILLEGAL DIRECT error 249
ILLEGAL QUANTITY error 40, 42, 52, 66, 86ff, 92, 97, 109, 112, 121ff, 129, 138ff, 146, 147, 161ff, 170, 171, 175ff, 249
immediate execution 4, 7, 9, 257
IN# statement 96, 223
index variable 55ff, 219, 225, 271
infinite loop 58
input 95, 223
numeric 100
Input Anything Routine 102
INPUT statement 16, 17, 97, 102, 223, 249, 294
input/output 93ff
string 99
INT function 39, 223, 291
integer constants 270
part 39, 223
variables 26, 27, 44, 58, 270, 275ff
Integer BASIC 260, 291
INVERSE statement 126, 128, 223, 226

J
JMP (Jump) instruction 173, 233
JSR (Jump to Subroutine)
instruction 173, 174, 246
K
keyboard 96, 258
keyword tokens 280ff
keywords 4

L
LEFT$ function 86, 100, 223, 249
LEFT-ARROW key 18, 19, 100, 241
LEN function 83, 85, 224
LET statement 215, 224
less than (<) 34
less than or equal to (<= or =<) 34
line feed character (CONTROL-J) 192, 193, 216, 255
line numbers 5ff, 50, 51, 64, 65, 70, 180, 220, 226, 232, 233, 251, 255, 267, 293, 294
LIST Command 7, 10, 224
LOAD Command 14, 110, 224, 298
LOG function 42, 224
logarithm, natural 42, 224
logical operators 35, 54
logical values 33, 36, 54
LOMEM: statement 177, 225, 250
loops 10, 55ff, 219, 225, 250, 270, 296
body 55
crossed 60
nested 59
low-resolution graphics 135, 216, 220, 221, 231, 234, 258, 261

M
machine language 172, 176, 177, 179, 216, 221, 233, 246
mask 174
MAT functions 296
memory allocation 25, 275
memory management 176
MID$ function 87, 100, 225, 249
minus sign (-) 36, 105
mixed graphics and text 119, 136, 138, 140, 141, 143, 146, 220, 260
Monitor program 16, 72, 155ff, 172, 173, 176, 177, 181
multidimensional array 80
multiple input 98
multiple statements per line 5
multiplication 32

N
natural logarithm 42, 224
nested loops 59
nested subroutines 62
NEW command 9, 30, 150, 177, 225
NEXT statement 55ff, 225, 271, 294
NEXT WITHOUT FOR error 10, 60, 249
NORMAL statement 126, 128, 226
NOT 35, 54
not equal to (<> or !=) 34
NOTTRACE command 181, 226
null character (CONTROL-@) 98, 100, 101, 105
null string 9, 12, 28, 30, 81, 82, 88, 97, 98, 100, 106, 251, 294
number formats 117
number sign(#) 96, 111, 180, 246
numeric constants 117, 283
numeric input 100

O
ON...GOSUB statement 65, 226, 249
ON...GOTO statement 51, 226, 249
on-screen edit 17
ONERR GOTO statement 68, 72, 226, 229, 247, 239, 264, 265
OPEN-APPLE key 110, 262
operators 31ff
arithmetic 30
logical 35, 54
precedence of 36
relational 33, 54
OR 34, 54
OUT OF DATA error 106, 250
OUT OF MEMORY error 60, 64, 177, 178, 250, 299
output 111
OVERFLOW error 90, 91, 250

P
parentheses 37, 250, 276
PD function 109, 227
PEEK function 68, 70, 110, 130, 131, 170, 177, 178, 180, 227, 247, 249, 253ff, 294
percent character (%) 26, 28
period (.) 105
PLOT statement 138, 227
plotting vector 150ff
plus sign (+) 36, 84, 105, 295
point of call 61, 64
pointer 275
<table>
<thead>
<tr>
<th>Statement</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>POKE</td>
<td>71, 72, 129ff, 136, 143, 149, 155, 156, 159, 170ff, 227, 249, 253ff, 294</td>
</tr>
<tr>
<td>POP</td>
<td>66, 227</td>
</tr>
<tr>
<td>POS function</td>
<td>125, 228</td>
</tr>
<tr>
<td>pound sign (#)</td>
<td>96</td>
</tr>
<tr>
<td>PR# statement</td>
<td>10, 111, 228</td>
</tr>
<tr>
<td>POKE statement</td>
<td>71, 72, 129ff, 136, 143, 149, 155, 156, 159, 170ff, 227, 249, 253ff, 294</td>
</tr>
<tr>
<td>PRINT statement</td>
<td>105, 113ff, 120, 121, 223, 226, 228, 231, 232, 254, 267</td>
</tr>
<tr>
<td>TAB used in</td>
<td>121ff</td>
</tr>
<tr>
<td>printer</td>
<td>10, 111</td>
</tr>
<tr>
<td>program</td>
<td>275</td>
</tr>
<tr>
<td>execution</td>
<td>16</td>
</tr>
<tr>
<td>layout</td>
<td>189</td>
</tr>
<tr>
<td>lines</td>
<td>3</td>
</tr>
<tr>
<td>planning</td>
<td>185</td>
</tr>
<tr>
<td>specification</td>
<td>185</td>
</tr>
<tr>
<td>prompt character (J)</td>
<td>4, 16, 119, 247</td>
</tr>
<tr>
<td>prompting message</td>
<td>97, 294</td>
</tr>
<tr>
<td>pure cursor moves</td>
<td>19</td>
</tr>
<tr>
<td>Q</td>
<td>question mark (?)</td>
</tr>
<tr>
<td>R</td>
<td>radians 40, 41, 44</td>
</tr>
<tr>
<td>RAM (random-access memory)</td>
<td>176, 179</td>
</tr>
<tr>
<td>random numbers</td>
<td>43, 229</td>
</tr>
<tr>
<td>READ statement</td>
<td>105, 108, 207, 217, 129, 250</td>
</tr>
<tr>
<td>real variables</td>
<td>25, 27, 44, 58, 270, 275-277</td>
</tr>
<tr>
<td>RECALL statement</td>
<td>110, 298</td>
</tr>
<tr>
<td>REDIM 'D ARRAY</td>
<td>79, 250</td>
</tr>
<tr>
<td>REENTER message</td>
<td>99, 100</td>
</tr>
<tr>
<td>relational operators</td>
<td>33, 54, 82</td>
</tr>
<tr>
<td>REM statement</td>
<td>7, 229, 267</td>
</tr>
<tr>
<td>reserved words</td>
<td>27, 245-246, 276</td>
</tr>
<tr>
<td>reset vector 16</td>
<td></td>
</tr>
<tr>
<td>restarting the system</td>
<td>96, 112, 176, 181</td>
</tr>
<tr>
<td>RESTORE statement</td>
<td>106, 108, 229, 250</td>
</tr>
<tr>
<td>Restoring Normal Error Handling</td>
<td>71</td>
</tr>
<tr>
<td>RESUME statement</td>
<td>69, 70, 229, 249, 265</td>
</tr>
<tr>
<td>return address</td>
<td>63, 66, 227</td>
</tr>
<tr>
<td>RETURN key</td>
<td>4, 6, 10, 13, 16, 18, 100, 104, 158, 165, 219, 241, 293</td>
</tr>
<tr>
<td>INPUT statement use</td>
<td>97, 98</td>
</tr>
<tr>
<td>RETURN statement</td>
<td>61ff, 220, 227, 251</td>
</tr>
<tr>
<td>RETURN WITHOUT GOSUB</td>
<td>error 64, 67, 251</td>
</tr>
<tr>
<td>right bracket (])</td>
<td>4, 16, 119, 247</td>
</tr>
<tr>
<td>RIGHT\ function</td>
<td>100, 229, 249</td>
</tr>
<tr>
<td>RIGHT-ARROW key</td>
<td>18, 19, 241</td>
</tr>
<tr>
<td>RND function</td>
<td>43, 229</td>
</tr>
<tr>
<td>RDT = statement</td>
<td>160, 164, 230</td>
</tr>
<tr>
<td>rotation 230</td>
<td></td>
</tr>
<tr>
<td>rounding 39</td>
<td></td>
</tr>
<tr>
<td>RTS (Return From Subroutine)</td>
<td>174</td>
</tr>
<tr>
<td>RUN Command</td>
<td>12, 14, 30, 108, 145, 150, 230, 294</td>
</tr>
<tr>
<td>S</td>
<td>SAVE Command 13, 131, 230, 297</td>
</tr>
<tr>
<td>SCAI.1E Command</td>
<td>13, 131, 230, 297</td>
</tr>
<tr>
<td>SCALE = statement</td>
<td>160, 163, 164, 230</td>
</tr>
<tr>
<td>scientific notation</td>
<td>43, 91, 118, 283</td>
</tr>
<tr>
<td>SCRN function</td>
<td>141, 231</td>
</tr>
<tr>
<td>scrolling</td>
<td>253</td>
</tr>
<tr>
<td>seeding</td>
<td>43</td>
</tr>
<tr>
<td>semicolon (:)</td>
<td>113ff, 122, 267, 269</td>
</tr>
<tr>
<td>SGN function</td>
<td>39, 231</td>
</tr>
<tr>
<td>shape definition</td>
<td>150</td>
</tr>
<tr>
<td>shape table(s)</td>
<td>150ff, 230, 231, 234, 299</td>
</tr>
<tr>
<td>index 153</td>
<td></td>
</tr>
<tr>
<td>loading 154ff</td>
<td></td>
</tr>
<tr>
<td>SHLOAD statement</td>
<td>110, 156, 158, 165, 231, 299</td>
</tr>
<tr>
<td>sign of a number</td>
<td>39, 231</td>
</tr>
<tr>
<td>simple variables</td>
<td>275-277</td>
</tr>
<tr>
<td>SIN function</td>
<td>40, 231</td>
</tr>
<tr>
<td>sine</td>
<td>40, 231</td>
</tr>
<tr>
<td>slash (/)</td>
<td>296</td>
</tr>
<tr>
<td>soft switches</td>
<td>253, 259</td>
</tr>
<tr>
<td>SOLID-APPLE key</td>
<td>110, 262</td>
</tr>
<tr>
<td>space bar</td>
<td>19, 21</td>
</tr>
<tr>
<td>space character</td>
<td>99, 101, 105, 231</td>
</tr>
<tr>
<td>SPC function</td>
<td>113, 120-121, 231, 249</td>
</tr>
<tr>
<td>speaker</td>
<td>130, 264</td>
</tr>
<tr>
<td>SPEED statement</td>
<td>128, 231</td>
</tr>
<tr>
<td>SQR function</td>
<td>40, 232, 249</td>
</tr>
<tr>
<td>square root</td>
<td>40, 232</td>
</tr>
<tr>
<td>statements 3, 223, 269</td>
<td>269</td>
</tr>
<tr>
<td>see also names of statements</td>
<td>57ff</td>
</tr>
<tr>
<td>step value 57ff</td>
<td></td>
</tr>
<tr>
<td>stepwise refinement</td>
<td>189</td>
</tr>
<tr>
<td>Term</td>
<td>Page Numbers</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>STOP statement 17, 73, 216</td>
<td></td>
</tr>
<tr>
<td>STR$ function 89, 232</td>
<td></td>
</tr>
<tr>
<td>string(s) 28, 81, 113, 229, 232, 233,</td>
<td></td>
</tr>
<tr>
<td>270, 275ff, 293, 295</td>
<td></td>
</tr>
<tr>
<td>comparison 82</td>
<td></td>
</tr>
<tr>
<td>constants 28, 81, 83</td>
<td></td>
</tr>
<tr>
<td>conversion 89</td>
<td></td>
</tr>
<tr>
<td>input 99</td>
<td></td>
</tr>
<tr>
<td>null 28</td>
<td></td>
</tr>
<tr>
<td>pointers 275-277</td>
<td></td>
</tr>
<tr>
<td>storage 179</td>
<td></td>
</tr>
<tr>
<td>variables 26, 28, 44, 83, 102, 104, 105</td>
<td></td>
</tr>
<tr>
<td>STRING TOO LONG error 84, 85, 114, 251</td>
<td></td>
</tr>
<tr>
<td>subroutine(s) 10, 61ff, 171, 229, 250,</td>
<td></td>
</tr>
<tr>
<td>269, 270, 276</td>
<td></td>
</tr>
<tr>
<td>call 61</td>
<td></td>
</tr>
<tr>
<td>execution 220</td>
<td></td>
</tr>
<tr>
<td>nested 62</td>
<td></td>
</tr>
<tr>
<td>subscripts 29, 77, 79</td>
<td></td>
</tr>
<tr>
<td>substrings 86, 295</td>
<td></td>
</tr>
<tr>
<td>subtraction 32, 36</td>
<td></td>
</tr>
<tr>
<td>syntax definitions 235ff</td>
<td></td>
</tr>
<tr>
<td>syntax error 13, 14, 54, 58, 105, 107</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>TAB function 113, 120, 121ff, 126, 181, 232, 249, 254</td>
<td></td>
</tr>
<tr>
<td>TAN function 41, 233</td>
<td></td>
</tr>
<tr>
<td>tangent 41, 233</td>
<td></td>
</tr>
<tr>
<td>tape cassette 13, 14, 110, 156, 158, 165, 228, 230, 231, 297ff</td>
<td></td>
</tr>
<tr>
<td>termination 218, 232</td>
<td></td>
</tr>
<tr>
<td>text 142, 253</td>
<td></td>
</tr>
<tr>
<td>window 115, 119ff, 129, 136, 143, 221, 253ff</td>
<td></td>
</tr>
<tr>
<td>TEXT statement 119, 136, 143, 233, 258</td>
<td></td>
</tr>
<tr>
<td>TRACE command 180, 181, 226, 233, 294</td>
<td></td>
</tr>
<tr>
<td>trigonometric functions 40-41</td>
<td></td>
</tr>
<tr>
<td>true 33ff</td>
<td></td>
</tr>
<tr>
<td>truncation 28, 39, 51, 65, 86, 88, 91, 117, 120ff, 283</td>
<td></td>
</tr>
<tr>
<td>TYPE MISMATCH error 87, 88, 251</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>unconditional branch 50, 220</td>
<td></td>
</tr>
<tr>
<td>UNDEF 'D FUNCTION error 251</td>
<td></td>
</tr>
<tr>
<td>UNDEF 'D STATEMENT error 12, 50, 51, 64, 251, 268</td>
<td></td>
</tr>
<tr>
<td>UP-ARROW key 18, 19, 241</td>
<td></td>
</tr>
<tr>
<td>user-defined function 44-45</td>
<td></td>
</tr>
<tr>
<td>USR function 172, 233</td>
<td></td>
</tr>
<tr>
<td>utility strobe 131, 261, 264</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VAL function 83, 86, 90, 102, 105, 107, 233</td>
<td></td>
</tr>
<tr>
<td>validation of data 187</td>
<td></td>
</tr>
<tr>
<td>values, logical 33, 54</td>
<td></td>
</tr>
<tr>
<td>variable(s) 25ff, 51, 97, 98, 177, 216, 268</td>
<td></td>
</tr>
<tr>
<td>argument 44</td>
<td></td>
</tr>
<tr>
<td>index 55, 57, 58, 60</td>
<td></td>
</tr>
<tr>
<td>integer 26, 27, 44, 58</td>
<td></td>
</tr>
<tr>
<td>name 26, 293</td>
<td></td>
</tr>
<tr>
<td>real 25, 27, 44, 58, 270, 275ff</td>
<td></td>
</tr>
<tr>
<td>string 26, 28, 44, 102, 105</td>
<td></td>
</tr>
<tr>
<td>VLIN statement 140, 234</td>
<td></td>
</tr>
<tr>
<td>VATAB statement 119, 120, 124, 181, 234, 256</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>WAIT statement 174, 234, 249</td>
<td></td>
</tr>
<tr>
<td>wraparound 4, 120, 122</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>XDRAW statement 151, 161ff, 230, 231, 234</td>
<td></td>
</tr>
<tr>
<td>XPLOT statement 246</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>zero page 278</td>
<td></td>
</tr>
</tbody>
</table>

**Cast of Characters**

" (double quotation marks) 28, 81, 99, 102, 270

# (number sign) 96, 111, 180, 246

$ (dollar sign) 26, 29, 82, 88, 251, 259

% (percent character) 26, 28

& (ampersand) 246

( ) (parentheses) 37, 250, 276

* (asterisk) 31, 32

+ (plus sign) 31, 36, 84, 105

, (comma) 98ff, 105, 113ff

- (minus sign) 31, 36, 105

: (period) 105

/ (slash) 31, 296

: (colon) 5, 98ff, 105, 106, 177, 192, 246, 267, 296, 301

; (semi-colon) 113ff, 122, 267, 269
< (less than) 34
<= or =< (less than or equal to) 34
= (equal sign) 30, 34, 44, 129, 137, 145, 163, 246
> (greater than) 34
>= or => (greater than or equal to) 34
<> or >>= (not equal to) 34
? (question mark) 97, 116, 294
] (right bracket) 4, 16, 119, 247
\ (backslash) 4, 18
^ (caret) 31
80-Column Text Card 4, 112ff, 119, 124, 125, 127, 222, 254, 287ff
Apple II

Applesoft BASIC
Quick Reference Card

Editing and Cursor Control

| LEFT-ARROW | Erase previous character |
| RIGHT-ARROW | Cancel input line |
| CONTROL-D | Erase character under cursor |

| ESC A | Move right; leave escape mode |
| ESC B | Move left; leave escape mode |
| ESC C | Move down; leave escape mode |
| ESC D | Move up; leave escape mode |

| ESC I | Move up; remain in escape mode |
| ESC J | Move left; remain in escape mode |
| ESC K | Move right; remain in escape mode |
| ESC M | Move down; remain in escape mode |

Operations on Whole Programs

| NEW | Erase current program, reset all variables |
| CLEAR | Reset all variables |
| LIST | Display current program |
| RUN | Execute program from beginning |
| RUN n | Execute program starting at line n |
| LOAD | Load program from tape |
| LOAD name | Load program name from disk |
| SAVE | Save current program on disk |
| SAVE name | Save current program on disk as name |

Interrupting and Resuming

| CONTROL-S | Suspend output (any key to resume) |
| CONTROL-C | Interrupt program execution |
| CONT | Continue execution after CONTROL-C, STOP, or END |
| CONTROL-R | Cancel program execution |

Statements and Lines

Lines typed without a line number are executed immediately; those with a line number are saved for later (deferred) execution.

| DEL n1+n2 | Delete from line n1 to line n2 |

Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>A7</td>
<td>+/-.9.99999999 E+37</td>
</tr>
<tr>
<td>Integer</td>
<td>ABZ</td>
<td>+/-.32767</td>
</tr>
<tr>
<td>String</td>
<td>A8</td>
<td>0 to 255 characters</td>
</tr>
</tbody>
</table>

where A is a letter, B is a letter or digit. Name may be more than two characters, but only first two are significant.

Interrupting and Resuming

| CONTROL-S | Suspend output (any key to resume) |
| CONTROL-C | Interrupt program execution |
| CONT | Continue execution after CONTROL-C, STOP, or END |
| CONTROL-R | Cancel program execution |

Statements and Lines

Lines typed without a line number are executed immediately; those with a line number are saved for later (deferred) execution.

| DEL n1+n2 | Delete from line n1 to line n2 |

Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>AB</td>
<td>+/-.9.99999999 E+37</td>
</tr>
<tr>
<td>Integer</td>
<td>ABZ</td>
<td>+/-.32767</td>
</tr>
<tr>
<td>String</td>
<td>A8</td>
<td>0 to 255 characters</td>
</tr>
</tbody>
</table>

where A is a letter, B is a letter or digit. Name may be more than two characters, but only first two are significant.

 índices  

<table>
<thead>
<tr>
<th>Control</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOTO n</td>
<td>Accept input from slot n</td>
</tr>
<tr>
<td>ON expr GOTO n1</td>
<td>Accept input from keyboard</td>
</tr>
<tr>
<td>RETURN</td>
<td>Prompt with string s, then read values into variables x, y, z, if s omitted, ? is used</td>
</tr>
<tr>
<td>IF cond</td>
<td>Read one character into variable c</td>
</tr>
<tr>
<td>THEN s1</td>
<td>Read values from DATA list</td>
</tr>
<tr>
<td>s2</td>
<td>into variables x, y, z</td>
</tr>
<tr>
<td>s3</td>
<td>omitted, 1 is understood</td>
</tr>
<tr>
<td>END</td>
<td>Read array a from tape</td>
</tr>
<tr>
<td>ON ERR GOTO n1</td>
<td>Read dial of hand control n</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Send output to slot n</td>
</tr>
<tr>
<td>GO TO n</td>
<td>Send output to display screen</td>
</tr>
<tr>
<td>LPRINT</td>
<td>Display or print values x, y, z</td>
</tr>
<tr>
<td>TEXT</td>
<td>Write array a to tape</td>
</tr>
<tr>
<td>HOME</td>
<td>Display text</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Clear screen and send cursor to top</td>
</tr>
<tr>
<td>MOVE</td>
<td>Start next item at cursor position</td>
</tr>
<tr>
<td>PRINT</td>
<td>Start next item at next tab position</td>
</tr>
<tr>
<td>DATA</td>
<td>Display or print x spaces (PRINT statement only)</td>
</tr>
<tr>
<td>DATA x</td>
<td>Move cursor to column x (PRINT statement only)</td>
</tr>
<tr>
<td>DATA y</td>
<td>Move cursor to column x</td>
</tr>
<tr>
<td>DATA z</td>
<td>Move cursor to line y</td>
</tr>
<tr>
<td>HLIB</td>
<td>Current horizontal cursor position</td>
</tr>
<tr>
<td>HLIB x</td>
<td>Display text in black-on-white</td>
</tr>
<tr>
<td>HLIB y</td>
<td>Display flashing text</td>
</tr>
<tr>
<td>HLIB z</td>
<td>Display text in white-on-black</td>
</tr>
<tr>
<td>POS</td>
<td>Set text display rate to x (0 minimum, 255 maximum)</td>
</tr>
</tbody>
</table>

Strings

| LEN | Concatenate strings |
| LENS | Length of string s |
| LEFT$ | Leftmost x characters of string s |
| MID$ | y characters beginning at position x in string s |
| RIGHT$ | Rightmost x characters of string s |
| STR$ | String representing numeric value x |
| VAL | Numeric value of string s |
| CHR$ | Character with ASCII code x |
| ASC | ASCII code for first character of string s |
Relational Operators

Yield value
Less than
Less than or equal to
Greater than
Greater than or equal to
Not equal to

Interpret 0 as false, nonzero as true. Yield value 0 if false, 1 if true.

Arithmetic Operators

Assign value to variable (LET optional)
Addition
Subtraction
Division
Exponentiation

Arithmetic Functions

Absolute value of x
Sign of x
Integer part of x
Square root of x
Sine of x radians
Cosine of x radians
Tangent of x radians
Arc tangent, in radians, of x
Exponential of x
Natural logarithm of x

If x > 0, generate random number between 0 and 1. If x = 0, repeat previous random number. If x < 0, begin new repeatable sequence of random numbers

Define function

Precedence of Operators

Parentheses
Signed arithmetic, logical "not"
Exponentiation
Multiplication, division
Addition, Subtraction
Relational operators

Arrays

Type       Typical Element
Real       A(x, y, z)
Integer    AB(x, y, z)
String     AB$(x, y, z)

where A is a letter, B is a letter or digit. Name may be more than two characters, but only first two are significant. Array size limited only by available memory.

DIM a(x, y, z) Define array a with maximum
subscripts x, y, z

Arithmetic Operators

=          Assign value to variable (LET optional)
+          Addition
-          Subtraction
*          Division
/          Exponentiation

Relational Operators

=          Equal to
<          Less than
<=         Less than or equal to
>          Greater than
>=         Greater than or equal to
<>         Not equal to

Columns numbered from 0 to 39; rows from 0 to 39 in mixed text and graphics, 0 to 47 in full-screen graphics.

Logical Operators

AND        Both true
OR         Either or both true
NOT        Is false

Columns numbered from 0 to 279; rows from 0 to 155 in mixed text and graphics, 0 to 191 in full-screen graphics.

Graphics

GR
COLOR = x
COLOR = x
COLOR = x
COLOR = x
COLOR = x
COLOR = x
COLOR = x
COLOR = x

Display low-resolution graphics
Set low-resolution display color to x
Plot single block at column x, row y
Draw horizontal line from column x1 to column x2 in row y
Draw vertical line from row y1 to row y2 in column x
Color on screen at column x, row y

Utility Statements

PEEK (addr)  Contents of memory location addr
POKE addr, x Store value x at memory location addr
CALL addr Execute machine-language subroutine starting at location addr
USR addr Execute user-supplied machine-language function with argument x
WAIT addr, m1, m2 Suspend execution until bit pattern specified by masks m1, m2 appears at location addr
HIMEM: addr Set highest memory address available for variable storage to addr
LOMEM: addr Set lowest memory address available for variable storage to addr
FRE (0)     Amount of available storage remaining

Display line number of each statement executed
Stop displaying line number of each statement executed

HGR
HGR2
HCOLOR = x
HPLDT x, y
HPLDT x1, y1, x2, y2, x3, y3
HPLDT x, y

Display high-resolution graphics, page 1; mixed text and graphics
Display high-resolution graphics, page 2; full-screen graphics
Set high-resolution display color to x
Plot single point at column x, row y
Draw high-resolution lines from column x1, row y1 to column x2, row y2 to column x3, row y3
Extend previous line to column x, row y

Columns numbered from 0 to 279; rows from 0 to 155 in mixed text and graphics, 0 to 191 in full-screen graphics.

SHLOAD
DRAW n AT
XDRAW n AT
SCALE = x
RDT = x

Load shape table from tape
Draw shape number n at column x, row y
Erase shape number n at column x, row y
Set scale factor for drawing shapes to x
Set rotation for drawing shapes to x