

# ATARI BASIC™

## **R e f e r e n c e** *Guide* For Experienced Programmers

**Learning BASIC is like learning any other language—it takes a little time and effort, but the rewards are great. This guide provides information about ATARI BASIC—a popular, powerful dialect—for those who are already familiar with the BASIC programming language.**

**This guide is intended for reference use only. It does not present comprehensive programming examples or tutorial information for the beginner. Both beginning**

**and experienced programmers should refer to the following sources for more information: YOUR ATARI COMPUTER: A GUIDE TO ATARI 400/800 PERSONAL COMPUTERS, by Ion Poole with Martin McNiff & Steven Cook; ATARI BASIC REFERENCE MANUAL; and INSIDE ATARI BASIC by Bill Carris.**

## Operator Precedence

Operations within the innermost set of parentheses are performed first and proceed out to the next level. When sets of parentheses are enclosed in another set, they are said to be "nested". Operations on the same nesting level are performed in the following order:

Highest precedence	<, >, =, <=, >=, <>	Relational operators used in string expressions have the same precedence and are performed from left to right.
	-	Unary minus (denotes a negative number)
	^	Exponentiation.
	* /	Multiplication and division have the same precedence level and are performed from left to right.
	+ , -	Addition and subtraction have the same precedence level and are performed from left to right.
	<, >, =, <=, >=, <>	Relational operations in numeric expressions have the same precedence level from left to right.
	NOT	Unary operator
	AND	Logical AND
Lowest precedence	OR	Logical OR

**(Allowable abbreviations in parenthesis.)**

The following words can be used as program statements, or as direct commands by typing them without a line number and pressing RETURN. These words may not be used as variable names.

## System Control

**BYE (B.)** — Exits from BASIC to SELF TEST MODE

**DOS** — Displays DOS menu (use with a disk drive, only).

**CSAVE (CS.)** — Saves program to Cassette.

**CLOAD** — Loads program from Cassette.

**SAVE (S.)** — Saves a BASIC program to an output device. Ex.: SAVE "D:MYFILE.BAS"

**LOAD (LO.)** — Loads a program from an input device. Ex.: LOAD "D:MYFILE.BAS"

**LIST (L.)** — Lists a program to screen or output device.

Ex.: List (Lists whole program)  
LIST 10 (lists line 10 on screen)  
LIST 10,20 (lists everything from line 10 to line 20)  
LIST "P:" (list to printer)  
LIST "P:",10,20 (lines 10-20 to printer)  
LIST "D:MYFILE.LST" (list to a disk file)  
LIST "D:MYFILE.LST",10,20 (list 10-20 to a disk file)  
LIST "C:" (List to cassette)

**ENTER (E.)** — Enters program from input device.

Ex.: ENTER "C:"  
ENTER "D:MYFILE.LST"

**NEW** — Clears program from memory.

**RUN** — Begins execution of a BASIC program. Program may be in memory or loaded from disk or tape. (Initializes variables to zero and un-dimensions arrays and strings.)

Ex.: RUN (executes program in memory)  
RUN "D:MYFILE.BAS" (LOADs program from disk and executes it)

**CONT** — Continues program execution after the break key has been pressed or the program has executed a STOP or END. If additional program statements are on the same line, they are not executed. The program continues execution with the next numbered line.

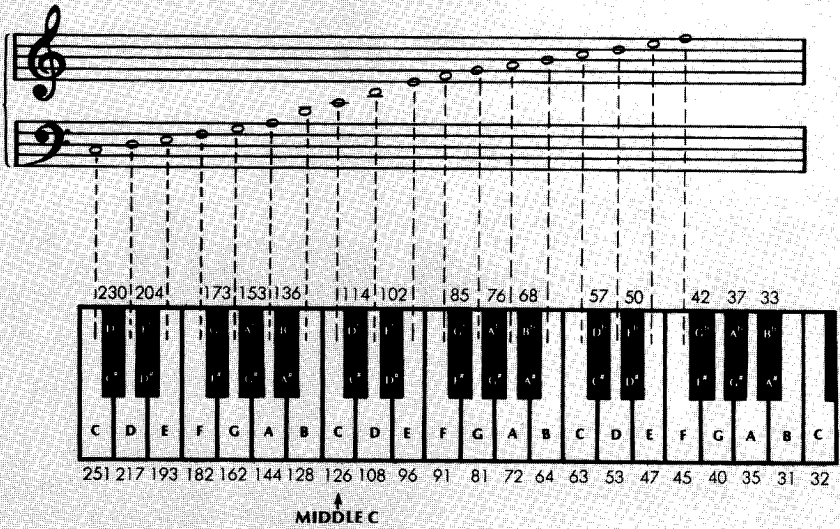
**SOUND (SO.)** — Sets one of four channels to produce sound through the TV speaker. The sound continues until another SOUND statement addresses the same channel, or an END, RUN or NEW statement is executed. Channels are programmed independently and may all be on at once. This statement must be followed by four values (numbers, variables or expressions).

Ex.: SOUND A,B,C,D where:  
A = Channel number (0-3)

B = Period (0-255) The larger the value, the lower the frequency. Frequency =  $31960 / (\text{PERIOD} + 1)$   
See chart for musical equivalents.

C = Distortion (0-14, even #'s only) Ten and 14 are "pure" tones. Other numbers result in other noises.

D = Volume (0-15) The larger the value, the louder the sound. 0 is off. If the total volume for all 4 voices exceeds 32, the speaker may "buzz".



**Relation of Piano Keyboard to Musical Scale**

**GOTO (G.)** — Program execution continues at the line number specified.  
Ex.: GOTO 30(execute program from line 30.)

Ex.: GOTO A + 10 (Legal, but can be hard to debug.)

**ON...GOTO** — Program execution continues at the line number indicated by an expression.

Ex.: ON A GOTO 10,300,50 (If A=1 then GOTO 10; if A=2 then GOTO 300; if A=3 then GOTO 50.)

**GOSUB (GOS.)** — Program execution continues at the line number specified. A RETURN statement will return control to the statement following the GOSUB.

Ex.: GOSUB 30 (execute subroutine at line 30)  
GOSUB A + 10 (Legal, but can be very hard to debug.)

**ON...GOSUB** — Program execution continues at the line number indicated by an expression. A RETURN will return control to the statement following the ON...GOSUB.

Ex.: ON A + 1 GOSUB 10,300,50  
(If A + 1 = 1 then GOSUB 10;  
if A + 1 = 2 then GOSUB 300;  
if A + 1 = 3 then GOSUB 50.)

**RETURN (RET.)** — Ends a subroutine and returns control to the statement immediately following the last GOSUB statement executed.

Ex.: RETURN

**FOR (F.)** — This statement sets up the starting and ending values of an index variable and the value to be added to it each time a FOR...NEXT loop executes. The value added is 1 unless specified otherwise by a STEP statement. A NEXT statement will cause the instructions between FOR and NEXT to repeat.

Ex.: FOR A = 1 TO 10 (A will start at 1, increase by 1 and stop at 10.)  
FOR A = 10 TO 1 STEP -2 (A negative STEP)

FOR A = B/T TO B\*T STEP X  
(Computed values are valid, too.)

**NEXT (N.)** — Ends a FOR...NEXT loop. Checks that the index value has not passed the end value, adds the step value to the index value and continues execution at the statement after the FOR. If the index passes the end value, goes to the statement after the NEXT.

Ex.: NEXT A (A is the index variable.)

**POP** — Removes return information stored about the last FOR or GOSUB statement executed. Useful for leaving a FOR-NEXT loop early, or leaving a subroutine without executing RETURN.

Ex.: POP

**IF...THEN** — The statement after the THEN is executed when the condition between IF and THEN is true. Otherwise goes to the next program line.

Ex.: IF A = B THEN GOTO 300  
IF A < B THEN PRINT  
"A < B":PRINT "HOW ABOUT  
THAT!":LET A = 5:GOTO 20  
IF A THEN PRINT "A is  
non-zero" (A non-zero expression  
is True.)

**TRAP (T.)** — On an error TRAP goes to the line specified. TRAP stays set until an error occurs or next TRAP statement. PEEK(195) returns the error number.

PEEK(187\*256 + PEEK(186)) returns the line number.

Ex.: TRAP 30 (On an error, go to line 30.)

TRAP 40000 (Line numbers greater than 32767 turn TRAP off.)

**STOP** — Stops program. Prints line number. Does not close files or turn off sound. Restart with CONT.

Ex.: STOP

**END** — Stops program, closes all open files, turns off sound.

**(I/O)**

**Device Names**

Each device in the ATARI family has a unique device name. Disk drives and the ATARI 850 Interface Module (RS232 handler) require a device number (1-4). Disk drives also require a file name. Device names must be enclosed in quotes or contained in string variables. Here are some examples:

- K: Keyboard. Input only.
- P: Printer. Output only.
- C: Cassette, Input and Output.
- S: Screen (TV). Output only.
- E: Screen Editor (keyboard and screen combined). Input and Output.
- R: RS232 Handler (ATARI 850 Interface Module).

- D: FILENAME.EXT File  
::FILENAME.EXT'' drive #1.
- D2: FILENAME.EXT Same file on drive #2.

Disk file names start with a letter and can be up to 8 characters long. The filename can end with an optional extension. (A period followed by 1-3 characters.) You can use any 3 letters or numbers you want. Some useful extensions are:

- .BAS = SAVED BASIC programs
- .LST = LISTED BASIC programs
- .DAT = data files
- .OBJ = Machine language (object) file
- .TXT = Text file

**I/O Statements**

**OPEN (O.)** — Prepares a device for input or output. IOCB numbers are 1-7. Uses the following codes:

TYPE	CODE	OPERATION
Input	4	Read only.
Output	8	Write only.
Update	12	Read and write
Append	9	Add to end of file
Directory	6	Disk drive directory only

- Ex.: OPEN #1,4,0,"K:" (Open keyboard for input on IOCB #1.)
- OPEN #2,8,0,"P:" (Open printer for output on IOCB #2.)
- OPEN #1,12,0,"D:MYFILE.DAT" (Open disk file "MYFILE.DAT" for update on IOCB #1.)
- OPEN #1,6,0,"D:\*.\*" (Open drive 1 for directory on IOCB #1.)

**CLOSE (CL.)** — Closes device after input or output operation and releases IOCB. May be executed when no device was opened. IOCB numbers are 1-7, as in OPEN.

- Ex.: CLOSE #1 (Close file open on IOCB #1 and release the IOCB.)

**INPUT (I.)** — Gets a line of characters from device. Line must be terminated by a RETURN character.

- Ex.: INPUT A (Get a number and put it in A)
- INPUT A,B,C (Get 3 numbers, separated by commas, and put them in A, B & C.)
- INPUT A\$ (Get a string of characters and put it in A\$, A\$ will not contain a RETURN character.)
- INPUT #1,A\$,B (Get a character string from device open on IOCB #1 and put in A\$ & B.)

**PRINT (PR.) or (?)** — Sends data to the screen or other device.

- Ex.: PRINT (Sends a blank line.)
- PRINT "The number is ";A (Prints text and number to screen.)
- PRINT "The number is ",A (The comma causes more space between string and number. Poke 201 with the number of spaces.)
- PRINT A\$; (Semicolon prevents RETURN from being sent at end of line.)

**PRINT #1;A\$** (Send A\$ to device open on IOCB #1.)

**LPRINT (LP.)** — Prints data on the printer. No open or close is necessary. Semicolon at end of line does not prevent RETURN from being sent.

Ex.: LPRINT (Send a blank line to printer.)

LPRINT A\$ (A\$ will be printed)

LPRINT A\$;B (A\$ and B will be on same line.)

LPRINT A\$,B (As above but comma causes more spaces to be put between items. POKE 201 with number of spaces.)

**GET** — Gets a single byte from device specified and puts it in the variable specified.

Ex.: GET #1,A (Gets a byte from device open on IOCB #1 and puts it in A.)

**PUT** — Puts the single byte in the variable to the device specified.

Ex.: PUT #1,A (Puts the byte in A to the device open on IOCB #1.)

**NOTE (NO.)** — Used with disk to determine location of next byte to be read or written.

Ex.: NOTE #1,SEC,BYTE (Will put sector number in SEC and byte number in BYTE. Refers to file open on IOCB #1.)

**POINT (P.)** — Used to tell DOS the location where the next byte is to be read from or written to.

Ex.: POINT #1,SEC,BYTE (Will point DOS at sector number SEC, byte number BYTE.)

**STATUS (ST.)** — Gets status for specified device. Status code returned can be found in the error message list.

Ex.: STATUS #1,A (Put status of device open on IOCB #1 in A.)

## Processing Statements

**LET** — Assigns values to numeric or string variables.

Ex.: LET A=B (value of B assigned to A)

LET A\$="HELLO"

A=B:A\$="HELLO"(LET can be omitted.)

**POKE** — Puts a number between 0 and 255 into a specified memory location between 0 and 65535.

Decimal values will be rounded.

Ex.: POKE 82,0 (Puts 0 into memory location 82.)

**DIM** — Reserves space in memory for strings and numeric arrays. Each character space reserved for a string takes one byte; each element in a numeric array, six bytes.

Ex.: DIM A\$(10) (a string variable)

DIM B(10) (a numerical array; B contains elements 0-10)

DIM B(10,10) (a 2 dimensional array)

DIM A\$(10),B(10) (Separate different items by commas.)

**COM** — Same as DIM.

**CLR** — Undimensions all arrays and strings:

Ex.: CLR

**DATA (D.)** — Creates a list of numbers and/or letters to be used by the READ statement below.

Ex.: DATA 1,2,3,4,A,B,C,D (A list of information to be read.)

**READ** — Reads the next item in a DATA statement and assigns it to a variable. When one DATA statement has been used, READ will get data from the next DATA statement in the program.

Ex: READ A (A will be the next number on the list in the DATA statement.)

READ A\$ (Works for strings, too.)

READ A,A\$,B,B\$ (Separate multiple items by commas.)

**RESTORE (RES.)** — Points READ at a DATA statement.

Ex.: RESTORE (Next byte will be first item from first DATA statement.)

RESTORE 10 (Next byte will be first item from DATA statement in line 10.)

**REM. (R.) or (. [space]).** — Allows remarks. BASIC ignores everything from REM to the end of the line.

Ex.: REM This is a remark statement!

#### THE ATARI HUE (SETCOLOR COMMAND) NUMBERS AND COLORS

COLORS	SETCOLOR (aexp2) NUMBERS
GRAY	0
LIGHT ORANGE (GOLD)	1
ORANGE	2
RED-ORANGE	3
PINK	4
PURPLE	5
PURPLE-BLUE	6
BLUE	7
BLUE	8
LIGHT BLUE	9
TURQUOISE	10
GREEN-BLUE	11
GREEN	12
YELLOW-GREEN	13
ORANGE-GREEN	14
LIGHT ORANGE	15

**Note:** Colors vary with type and adjustment of TV or monitor used.

#### TABLE OF SETCOLOR "DEFAULT" COLORS\*

Setcolor (Color Register)	Defaults To Color	Luminance	Actual Color
0	2	8	ORANGE
1	12	10	GREEN
2	9	4	DARK BLUE
3	4	6	PINK OR RED
4	0	0	BLACK

\*"DEFAULT" occurs if no SETCOLOR statement is used.

**Note:** Colors may vary depending upon the television monitor type, condition, and adjustment.



TABLE OF MODES AND SCREEN FORMATS

SCREEN FORMAT		Columns	Rows— Split Screen	Rows— Full Screen	Number of Colors	RAM Required (Bytes)	
Graphics Mode	Mode Type					Split	Full
0	TEXT	40	—	24	1-1/2		992
1	TEXT	20	20	24	5	674	672
2	TEXT	20	10	12	5	424	420
3	GRAPHICS	40	20	24	4	434	432
4	GRAPHICS	80	40	48	2	694	696
5	GRAPHICS	80	40	48	4	1174	1176
6	GRAPHICS	160	80	96	2	2174	2184
7	GRAPHICS	160	80	96	4	4190	4200
8	GRAPHICS	320	160	192	1-1/2	8112	8138
9	GRAPHICS	80	—	192	1		8138
10	GRAPHICS	80	—	192	9		8138
11	GRAPHICS	80	—	192	16		8138
12	GRAPHICS	40	20	24	5	1154	1152
13	GRAPHICS	40	10	12	5	664	660
14	GRAPHICS	160	160	192	2	4270	4296
15	GRAPHICS	160	160	192	4	8112	8138

## Graphics

**GRAPHICS (GR.)** — Selects graphics mode, Mode + 16 selects a full screen (no text window). Mode + 32 does not clear screen. Modes 0-15 available.

Ex.: GRAPHICS 8 (Graphics mode 8 with text window.)  
 GRAPHICS 8+16 (Mode 8, full screen)  
 GRAPHICS 8+32 (Won't clear screen)  
 GRAPHICS 8+16+32 (both options combined)

**SETCOLOR (SE.)** — Sets the hue and luminance of the chosen color register. Register number is not the same as with COLOR command.

Ex.: SETCOLOR 1,2,4 (Set reg. 1 to hue 2 and luminance 4.)  
 Registers 0-3  
 Hues 0-15  
 Luminances 0-14, even numbers only.

**COLOR (C.)** — In map modes (3-11) selects a color register to use for PLOT. Register here is not the same as the register in SETCOLOR.

Ex.: COLOR 2 (Selects color register 2 in modes 0-2, selects ASCII character whose value is 2 for PLOT.)

**PLOT (PL.)** — Puts a single point or character on the screen at a specified location.

Ex: PLOT X,Y (X and Y must be positive coordinates.)

**POSITION (POS.)** — Selects a screen position, but nothing is plotted. Useful for positioning text with PRINT.

Ex: POSITION X,Y

**LOCATE (LOC.)** — Retrieves data stored at a specified screen location. Gets characters in modes 0-2, color numbers in modes 3-11.

Ex: LOCATE X,Y,D (Moves to X, Y and puts data in D.)

**DRAWTO (DR.)** — Draws a line between the last position of the cursor and the specified X & Y coordinates. The cursor ends up at the new coordinates.

Ex.: DRAWTO X,Y (draws a line to point X,Y from the current position of the cursor.)

A function takes one or more values and returns another value. Values may be strings or numbers. The examples show A (a numeric variable) or A\$ (a string variable) as being equal to the function, but a function can be used almost anywhere you would use a value (including in another function).

**Arithmetic Functions**

**ABS** — Returns absolute (unsigned) value of a number.  
Ex.:  $A = \text{ABS}(B)$

**CLOG** — Returns the logarithm to the base 10 (common log).  
Ex.:  $A = \text{CLOG}(B)$

**EXP** — Returns the value of e (approximately 2.718) raised to the power specified. In some cases, EXP is accurate only to 6 significant digits. This is the reciprocal of LOG.  
Ex.:  $A = \text{EXP}(B)$

**INT** — Returns the greatest integer less than or equal to a value.  
Ex.:  $A = \text{INT}(B)$

**LOG** — Returns the natural logarithm of a value. This is the reciprocal of EXP.

**RND** — Returns a random number between 0 and 1. Never returns a 1. Value makes no difference.  
Ex.:  $A = \text{RND}(0)$  (A = a number greater than or equal to 0 and less than 1.)  
 $A = \text{RND}(0) * 8$  (A = a number greater than or equal to 0 and less than 8.)

**SGN** — Returns a -1 if value is negative, a zero if it's 0 and a 1 if it's positive.  
Ex.:  $A = \text{SGN}(A)$

**SQR** — Returns the positive square root of a positive value.  
Ex.:  $A = \text{SQR}(B)$

**Trigonometric Functions**

**ATN** — Returns the arctangent of a value in radians or degrees.  
Ex.:  $A = \text{ATN}(B)$

**SIN** — Returns the sine of the value.  
Ex.:  $A = \text{SIN}(B)$

**COS** — Returns the cosine of a value.  
Ex.:  $A = \text{COS}(B)$

**DEG** — All subsequent trig functions will be in degrees.  
Ex.: DEG

**RAD** — All subsequent trig functions will be in radians. The computer assumes you want radians unless you specify DEG (degrees).

**Special Purpose Functions**

**ADR** — Returns the decimal memory address of the beginning of a string.  
Ex.:  $A = \text{ADR}(B\$)$   
 $A = \text{ADR}(\text{"THIS STRING"})$

**FRE** — Returns the number of bytes of user RAM left. (0) is a required "dummy" variable.  
Ex.:  $A = \text{FRE}(0)$   
PRINT FRE(0) (This will tell you how much memory is left.)

**PEEK** — Returns the number stored at a specific memory location. The address must be a number between 0 and 65535. The number returned will be between 0 and 255.  
Ex.:  $A = \text{PEEK}(B)$

**USR** — Returns the results of a machine language subroutine. The first value is the address of a subroutine in memory. The other values will be put on the stack (hi byte first) for use by the subroutine. Note that USR leaves the number of parameters (not counting the address) on the top of the stack.  
Ex.:  $A = \text{USR}(B,C,D)$  (Will call routine at B and pass C and D to it.)

## String Functions

**ASC** — Returns the ATASCII code number for the first character of a string.

Ex.: A=ASC("A") (A will be 65.)  
A=ASC(B\$) (String variables can be used.)

**CHR\$** — Returns the character represented by the ATASCII code number specified. Reciprocal of ASC.  
Ex.: A\$=CHR\$(65) (A\$ will be "A".)

**LEN** — Returns a number that represents the length of a string variable.  
Ex.: A=LEN(A\$)

**STR\$** — Returns a string representing a specified value. (Translates a number into a string.)  
Ex.: A\$=STR\$(65) (A\$ will equal "65" which is not a number but a string.)

**VAL** — Returns a number representing a specific string. (Translates a string into a number.)  
Ex.: A=VAL("100") (A will equal the number 100.)

## String Manipulation

ATARI BASIC does not use a string array format for manipulating strings. A powerful mid-string command allows such things as concatenation of strings and other string handling.

Examples:

SUBSTRINGS

```
50 A$ = "DAVEMARKGRETCHEN"  
60 B$ = A$(9,16) (B$ is  
"GRETCHEN")
```

CONCATENATION

```
50 A$ = "HI"  
60 B$ = "FRED"  
70 A$(LEN(A$)+1) = B$ (A$ is "HI  
FRED")
```

SEARCHING A STRING

```
50 FOR I=1 TO LEN(A$)  
60 IF A$(I,I) = "E" THEN PRINT  
"AN E!"  
70 NEXT I
```

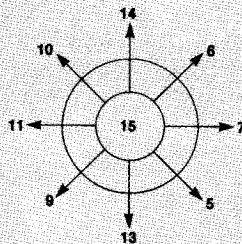
## Game Controller Functions

**PADDLE** — Returns the position of a specific paddle controller. The paddles are numbered 0-3 from front to back. Number returned is between 1 and 228, increasing as the knob is turned left (counterclockwise).  
Ex.: A=PADDLE (0)

**PTRIG** — Returns a 0 if a specific paddle trigger is pressed and a 1 if it isn't. The paddles are numbered 0-3 from front to back.  
Ex.: A=PTRIG(0)

**STICK** — Returns the status of a specific joystick controller. The joysticks are numbered 0-1 from front to back. See diagram for details.  
Ex.: A=STICK(0)

**STRIG** — Returns a 0 if a specific joystick trigger button is pressed and a 1 if it isn't. The joysticks are numbered 0 and 1.  
Ex.: A=STRIG(0)



**ESC** Causes next key pressed to be displayed as an international character or a graphics character, and executed when it's printed to the screen.

**BREAK** Causes a BASIC program to stop.

**SYSTEM RESET** Stops a running program, returns the screen to graphics mode 0, clears the screen, and sets variables to their default settings. Does not erase your program.

**SET-CLR-TAB** Moves the cursor to the next preset tab stop.

**SHIFT SET-CLR-TAB** Clears a tab.

**CTRL SET-CLR-TAB** Sets a tab.

**CTRL 1** Stops and starts screen display scrolling.

**CTRL 2** Sounds buzzer.

**CTRL 3** Indicates end of file.

**SHIFT INSERT** Inserts a line.

**CTRL INSERT** Inserts a character.

**DELETE BACK S** Deletes character to the left of cursor and moves cursor into the empty position.

**SHIFT DELETE BACK S** Deletes a line.

**CTRL DELETE BACK S** Deletes character at the cursor and moves remainder of line to fill in the empty position.

**SHIFT CLEAR** or **CTRL CLEAR** Clears screen.

**CTRL UP ARROW** Moves cursor up.

**CTRL DOWN ARROW** Moves cursor down.

**CTRL LEFT ARROW** Moves cursor left.

**CTRL RIGHT ARROW** Moves cursor right.

## ERROR

### CODE ERROR CODE MESSAGE

<b>2</b>	Memory Insufficient
<b>3</b>	Value Error
<b>4</b>	Too Many Variables
<b>5</b>	String Length Error
<b>6</b>	Out of Data Error
<b>7</b>	Number greater than 32767
<b>8</b>	Input Statement Error
<b>9</b>	Array or String <b>DIM</b> Error
<b>10</b>	Argument Stack Overflow
<b>11</b>	Floating Point Overflow/ Underflow Error
<b>12</b>	Line Not Found
<b>13</b>	No Matching <b>FOR</b> Statement
<b>14</b>	Line Too Long Error
<b>15</b>	<b>GOSUB</b> or <b>FOR</b> Line Deleted
<b>16</b>	<b>RETURN</b> Error
<b>17</b>	Syntax Error
<b>18</b>	Invalid String Character

**Note:** The following are INPUT/OUTPUT errors that result during the use of disk drives, printers, or other accessory devices. Further information is provided with the auxiliary hardware.

<b>19</b>	LOAD program Too Long
<b>20</b>	Device Number Larger
<b>21</b>	LOAD File Error
<b>128</b>	BREAK Abort
<b>129</b>	IOCB

<b>130</b>	Nonexistent Device
<b>131</b>	IOCB Write Only
<b>132</b>	Invalid Handler Command
<b>133</b>	Device or File not Open
<b>134</b>	BAD IOCB Number
<b>135</b>	IOCB Read Only Error
<b>136</b>	EOF
<b>137</b>	Truncated Record
<b>138</b>	Device Timeout
<b>139</b>	Device NAK
<b>140</b>	Serial Bus
<b>141</b>	Cursor Out of Range
<b>142</b>	Serial Bus Data Frame Overrun
<b>143</b>	Serial Bus Data Frame Checksum Error
<b>144</b>	Device Done Error
<b>145</b>	Bad Screen Mode Error
<b>146</b>	Function Not Implemented
<b>147</b>	Insufficient Screen RAM
<b>160</b>	Drive Number Error
<b>161</b>	Too many OPEN Files
<b>162</b>	Disk Full
<b>163</b>	Unrecoverable System Data I/O Error
<b>164</b>	File Number Mismatch
<b>165</b>	File Name Error
<b>166</b>	POINT Data Length Error
<b>167</b>	File Locked
<b>168</b>	Invalid Device Command
<b>169</b>	Directory Full
<b>170</b>	File Not Found
<b>171</b>	POINT Invalid
<b>172</b>	Illegal Append
<b>173</b>	Bad Format

ABS .....	10	LPRINT .....	7
ADR .....	10	NEW .....	3
AND .....	2	NEXT .....	5
ASC .....	11	NOT .....	2
ATN .....	10	NOTE .....	7
BYE .....	3	ON .....	5
CLOAD .....	3	OPEN .....	6
CHR\$ .....	11	OPERATORS ..	2
CLOG .....	10	OR .....	2
CLOSE .....	6	PADDLE .....	11
CLR .....	7	PEEK .....	10
COLOR .....	9	PLOT .....	9
COM .....	7	POINT .....	7
CONT .....	3	POKE .....	7
COS .....	10	POP .....	5
CSAVE .....	3	POSITION .....	9
DATA .....	7	PRINT .....	6
DEG .....	10	PTRIG .....	11
DIM .....	7	PUT .....	7
DOS .....	3	RAD .....	10
DRAWTO .....	9	READ .....	8
EDITING .....	12	REM .....	8
END .....	5	RESTORE .....	8
ENTER .....	3	RETURN .....	5
E R R O R		RND .....	10
MESSAGES .....	13	RUN .....	3
EXP .....	10	SAVE .....	3
FOR .....	5	SETCOLOR .....	9
FRE .....	10	SGN .....	10
GET .....	7	SIN .....	10
GOSUB .....	5	SOUND .....	4
GOTO .....	5	SPECIAL	
GRAPHICS .....	9	FUNCTION	
IF .....	5	KEYS .....	12
INPUT .....	6	SQR .....	10
INT .....	10	STATUS .....	7
LEN .....	11	STEP .....	5
LET .....	7	STICK .....	11
LIST .....	3	STRIG .....	11
LOAD .....	3	STOP .....	5
LOCATE .....	9	STR\$ .....	11
LOG .....	10	THEN .....	5
		TO .....	5
TRAP .....	5		
USR .....	10		
VAL .....	11		



