<file number> is an integer expression whose value is between 1 and 15. The number is associated with the file as long as it is OPEN and refers other disk I/O statements to the file.

<filename> is a string expression containing a name that conforms to your operating system's rules for disk filenames.

<reclen> is an integer expression which, if included, sets the record length for random files. The default record length is 128 bytes.

Note A file can be OPENed for sequential input or random access on more than one file number at a time. A file may be OPENed for output, however, on only one file number at a time.

Example

```
10 OPEN "I",2,"INVEN"
```

See also "Disk File Handling," in the *Microsoft BASIC User's Guide.*

2.46 OPTION BASE

Syntax

```
OPTION BASE n
```

where n is 1 or 0

Purpose To declare the minimum value for array subscripts.

Remarks The default base is 0. If the statement

```
OPTION BASE 1
```

is executed, the lowest value an array subscript may have is 1.

Example

```
OPTION BASE 1
```

2.47 OUT

Syntax OUT I,J

where I and J are integer expressions in the range 0 to 255.

Purpose To send a byte to a machine output port.

Remarks The integer expression I is the port number. The integer expression J is the data to be transmitted.

Example

```
100 OUT 32,100
```

2.48 POKE

Syntax POKE I,J

where I and J are integer expressions.

Purpose To write a byte into a memory location.

Remarks I and J are integer expressions. The expression I represents the address of the memory location and J is the data byte. I must be in the range -32768 to 65535. (For interpretation of negative values of I, see "VARPTR," Section 3.43.)

The complementary function to POKE is PEEK. The argument to PEEK is an address from which a byte is to be read. See Section 3.28.

POKE and PEEK are useful for storing data efficiently, loading assembly language subroutines, and passing arguments and results to and from assembly language subroutines.

Example

```
10 POKE &H5A00,&HFF
```

2.49 PRINT

Syntax PRINT [<list of expressions>]

Purpose To output data at the terminal.

Remarks If <list of expressions> is omitted, a blank line is printed. If <list of expressions> is included, the values of the expressions are printed at the terminal. The expressions in the list may be numeric and/or string expressions. (Strings must be enclosed in quotation marks.)

Print Positions

The position of each printed item is determined by the punctuation used to separate the items in the list. Microsoft BASIC divides the line into print zones of 14 spaces each. In the list of expressions, a comma causes the next value to be printed at the beginning of the next zone. A semicolon causes the next value to be printed immediately after the last value. Typing one or more spaces between expressions has the same effect as typing a semicolon.

If a comma or a semicolon terminates the list of expressions, the next PRINT statement begins printing on the same line, spacing accordingly. If the list of expressions terminates without a comma or a semicolon, a carriage return is printed at the end of the line. If the printed line is longer than the terminal width, Microsoft BASIC goes to the next physical line and continues printing.

Printed numbers are always followed by a space. Positive numbers are preceded by a space. Negative numbers are preceded by a minus sign. Single precision numbers that can be represented with 6 or fewer digits in the unscaled format no less accurately than they can be represented in the scaled format are output using the unscaled format. For example, 1E-7 is output as .0000001 and 1E-8 is output as 1E-08. Double precision numbers that

can be represented with 16 or fewer digits in the unscaled format no less accurately than they can be represented in the scaled format are output using the unscaled format. For example, 1D-15 is output as .0000000000000001 and 1D-16 is output as 1D-16.

A question mark may be used in place of the word PRINT in a PRINT statement.

Example 1

```
10 X=5
20 PRINT X+5,X-5,X*(-5),X^5
30 END
RUN
 10            0            -25           3125
Ok
```

In this example, the commas in the PRINT statement cause each value to be printed at the beginning of the next print zone.

Example 2

```
LIST
10 INPUT X
20 PRINT X "SQUARED IS" X^2 "AND";
30 PRINT X "CUBED IS" X^3
40 PRINT
50 GOTO 10
Ok
RUN
?  9
 9 SQUARED IS 81 AND 9 CUBED IS 729

?  21
 21 SQUARED IS 441 AND 21 CUBED IS 9261

?
```

In this example, the semicolon at the end of line 20 causes both PRINT statements to be printed on the same line. Line 40 causes a blank line to be printed before the next prompt.

Example 3

```
10 FOR X = 1 TO 5
20 J = J + 5
30 K = K + 10
40 ?J;K;
50 NEXT X
Ok
RUN
 5   10   10   20   15   30   20   40   25   50
Ok
```

In this example, the semicolons in the PRINT statement cause each value to be printed immediately after the preceding value. (Don't forget, a number is always followed by a space, and positive numbers are preceded by a space.) In line 40, a question mark is used instead of the word PRINT.

2.50 PRINT USING

Syntax

PRINT USING <string exp>;<list of expressions>

Purpose

To print strings or numbers using a specified format.

Remarks and Examples

<list of expressions> is comprised of the string expressions or numeric expressions that are to be printed, separated by semicolons. <string exp> is a string literal (or variable) comprised of special formatting characters. These formatting characters (see below) determine the field and the format of the printed strings or numbers.

String Fields

When PRINT USING is used to print strings, one of three formatting characters may be used to format the string field:

"!" Specifies that only the first character in the given string is to be printed.

"\n spaces\" Specifies that 2+n characters from the string are to be printed. If the backslashes are typed with no spaces, two characters will be printed; with one space, three characters will be printed, and so on. If the string is longer than the field, the extra characters are ignored. If the field is longer than the string, the string will be left-justified in the field and padded with spaces on the right.

Example

```
10 A$ = "LOOK":B$ = "OUT"
30 PRINT USING "!";A$;B$
40 PRINT USING "\ \";A$;B$
50 PRINT USING "\ \";A$;B$;"!!"
RUN
LO
LOOKOUT
LOOK OUT !!
```

"&" Specifies a variable length string field. When the field is specified with "&", the string is output without modification.

Example

```
10 A$ = "LOOK":B$ = "OUT"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
LOUT
```

Numeric Fields

When PRINT USING is used to print numbers, the following special characters may be used to format the numeric field:

A number sign is used to represent each digit position. Digit positions are always filled. If the number to be printed has fewer digits than positions specified, the number will be right-justified (preceded by spaces) in the field.

A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be printed (as 0, if necessary). Numbers are rounded as necessary.

```
PRINT USING "##.##";.78
  0.78

PRINT USING "###.##";987.654
987.65

PRINT USING "##.##   ";10.2,5.3,66.789,.234
10.20   5.30   66.79   0.23
```

In the last example, three spaces were inserted at the end of the format string to separate the printed values on the line.

+ A plus sign at the beginning or end of the format string will cause the sign of the number (plus or minus) to be printed before or after the number.

- A minus sign at the end of the format field will cause negative numbers to be printed with a trailing minus sign.

```
PRINT USING "+##.##   ";-68.95,2.4,55.6,-.9
-68.95   +2.40   +55.60   -0.90

PRINT USING "##.##-   ";-68.95,22.449,-7.01
68.95-   22.45   7.01-
```

****** A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits.

```
PRINT USING "**#.#   ";12.39,-0.9,765.1
*12.4   *-0.9   765.1
```

$$

A double dollar sign causes a dollar sign to be printed to the immediate left of the formatted number. The $$ specifies two more digit positions, one of which is the dollar sign. The exponential format cannot be used with $$. Negative numbers cannot be used unless the minus sign trails to the right.

```
PRINT USING "$$###.##";456.78
  $456.78
```

****$**

The **$ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be asterisk-filled and a dollar sign will be printed before the number. **$ specifies three more digit positions, one of which is the dollar sign.

```
PRINT USING "**$##.##";2.34
**$2.34
```

A comma that is to the left of the decimal point in a formatting string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma that is at the end of the format string is printed as part of the string. A comma specifies another digit position. The comma has no effect if used with the exponential (^^^^) format.

```
PRINT USING "####,.##";1234.5
1,234.50

PRINT USING "####.##,";1234.5
1234.50,
```

^^^^

Four carets (or up-arrows) may be placed after the digit position characters to specify exponential format. The four carets allow space for E+xx to be printed. Any decimal point position may be specified. The significant digits are left-justified, and the exponent is adjusted. Unless a leading + or trailing + or - is specified, one digit position will be used to the left of the decimal point to print a space or a minus sign.

```
PRINT USING "##.##^^^^";234.56
 2.35E+02

PRINT USING ".####^^^^-";888888
.8889E+06

PRINT USING "+.##^^^^";123
+.12E+03
```

_ An underscore in the format string causes the next character to be output as a literal character.

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

The literal character itself may be an underscore by placing "__" in the format string.

% If the number to be printed is larger than the specified numeric field, a percent sign is printed in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

```
PRINT USING "##.##";111.22
%111.22

PRINT USING ".##";.999
%1.00
```

If the number of digits specified exceeds 24, an "Illegal function call" error will result.

2.51 PRINT# and PRINT# USING

Syntax PRINT#<file number>,[USING <string exp>;]
<list of expressions>

Purpose To write data to a sequential disk file.

Remarks <file number> is the number that was used when

the file was OPENed for output. <string exp> is comprised of formatting characters as described in Section 2.50, "PRINT USING." The expressions in <list of expressions> are the numeric and/or string expressions that will be written to the file.

PRINT# does not compress data on the disk. An image of the data is written to the disk, just as it would be displayed on the terminal screen with a PRINT statement. For this reason, care should be taken to delimit the data on the disk, so that it will be input correctly from the disk.

In the list of expressions, numeric expressions should be delimited by semicolons. For example:

```
PRINT#1,A;B;C;X;Y;Z
```

(If commas are used as delimiters, the extra blanks that are inserted between print fields will also be written to the disk.)

String expressions must be separated by semicolons in the list. To format the string expressions correctly on the disk, use explicit delimiters in the list of expressions.

For example, let A$="CAMERA" and B$= "93604-1". The statement

```
PRINT#1,A$;B$
```

would write CAMERA93604-1 to the disk. Because there are no delimiters, this could not be input as two separate strings. To correct the problem, insert explicit delimiters into the PRINT# statement as follows:

```
PRINT#1,A$;",";B$
```

The image written to disk is

```
CAMERA,93604-1
```

which can be read back into two string variables.

If the strings themselves contain commas, semicolons, significant leading blanks, carriage returns, or line feeds, write them to disk surrounded by explicit quotation marks, CHR$(34).

For example, let A$="CAMERA, AUTOMATIC" and B$=" 93604-1". The statement

```
PRINT#1,A$;B$
```

would write the following image to disk:

```
CAMERA, AUTOMATIC  93604-1
```

And the statement

```
INPUT#1,A$,B$
```

would input "CAMERA" to A$ and "AUTOMATIC 93604-1" to B$. To separate these strings properly on the disk, write double quotation marks to the disk image using CHR$(34). The statement

```
PRINT#1,CHR$(34);A$;CHR$(34);CHR$(34);B$;
CHR$(34)
```

writes the following image to disk:

```
"CAMERA, AUTOMATIC""  93604-1"
```

And the statement

```
INPUT#1,A$,B$
```

would input "CAMERA, AUTOMATIC" to A$ and " 93604-1" to B$.

The PRINT# statement may also be used with the USING option to control the format of the disk file. For example:

```
PRINT#1,USING"$$###.##,";J;K;L
```

For more examples using PRINT#, see "Disk File Handling," in the *Microsoft BASIC User's Guide.*

See also "WRITE#," Section 2.68.

2.52 PUT

Syntax PUT [#]<file number>[,<record number>]

Purpose To write a record from a random buffer to a random disk file.

Remarks <file number> is the number under which the file was OPENed. If <record number> is omitted, the record will assume the next available record number (after the last PUT). The largest possible record number is 32,767. The smallest record number is 1.

Example See "Disk File Handling," in the *Microsoft BASIC User's Guide.*

Note PRINT#, PRINT# USING, and WRITE# may be used to put characters in the random file buffer before executing a PUT statement.

In the case of WRITE#, Microsoft BASIC pads the buffer with spaces up to the carriage return. Any attempt to read or write past the end of the buffer causes a "Field overflow" error.

2.53 RANDOMIZE

Syntax RANDOMIZE [<expression>]

Purpose To reseed the random number generator.

Remarks If <expression> is omitted, Microsoft BASIC suspends program execution and asks for a value by printing

Random Number Seed (-32768 to 32767)?

before executing RANDOMIZE.

If the random number generator is not reseeded, the RND function returns the same sequence of random numbers each time the program is RUN. To change the sequence of random numbers every time the program is RUN, place a RANDOMIZE statement at the beginning of the program and change the argument with each RUN.

Example

```
10 RANDOMIZE
20 FOR I = 1 TO 5
30 PRINT RND;
40 NEXT I
RUN
Random Number Seed (-32768 to 32767)? 3
     (user types 3)

 .88598 .484668 .586328 .119426 .709225
Ok
RUN
Random Number Seed (-32768 to 32767)? 4
     (user types 4 for new sequence)

 .803506 .162462 .929364 .292443 .322921
Ok
RUN
Random Number Seed (-32768 to 32767)? 3
     (same sequence as first RUN)

 .88598 .484668 .586328 .119426 .709225
Ok
```

2.54 READ

Syntax READ <list of variables>

Purpose To read values from a DATA statement and assign them to variables. (See "DATA," Section 2.10.)

Remarks A READ statement must always be used in conjunction with a DATA statement. READ statements assign variables to DATA statement values on a one-to-one basis. READ statement variables may be numeric or string, and the values read must agree with the variable types specified. If they do not agree, a "Syntax error" will result.

A single READ statement may access one or more DATA statements (they will be accessed in order), or several READ statements may access the same DATA statement. If the number of variables in <list of variables> exceeds the number of elements in the DATA statement(s), an "Out of data" error message is printed. If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements will begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

To reread DATA statements from the start, use the RESTORE statement (see "RESTORE," Section 2.57).

Example 1

```
.
.
.
80 FOR I = 1 TO 10
90 READ A(I)
100 NEXT I
110 DATA 3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
.
.
.
```

This program segment READs the values from the DATA statements into the array A. After execution, the value of A(1) will be 3.08, and so on.

Example 2

```
LIST
10 PRINT "CITY", "STATE", " ZIP"
20 READ C$,S$,Z
30 DATA "DENVER,", COLORADO, 80211
40 PRINT C$,S$,Z
Ok
RUN
CITY            STATE           ZIP
DENVER,         COLORADO        80211
Ok
```

This program READs string and numeric data from the DATA statement in line 30.

2.55 REM

Syntax REM <remark>

Purpose To allow explanatory remarks to be inserted in a program.

Remarks REM statements are not executed but are output exactly as entered when the program is listed.

REM statements may be branched into from a GOTO or GOSUB statement. Execution will continue with the first executable statement after the REM statement.

Remarks may be added to the end of a line by preceding the remark with a single quotation mark instead of :REM.

Important Do not use this in a data statement, because it would be considered legal data.

Example

```
.
.
.
120 REM CALCULATE AVERAGE VELOCITY
130 FOR I = 1 TO 20
140 SUM = SUM + V(I)
.
.
.
```

or

```
.
.
.
120 FOR I = 1 TO 20      'CALCULATE AVERAGE
VELOCITY
130 SUM = SUM + V(I)
140 NEXT I
.
.
.
```

2.56 RENUM

Syntax RENUM [[<new number>][,[<old number>][,<increment>]]]

Purpose To renumber program lines.

Remarks <new number> is the first line number to be used in the new sequence. The default is 10. <old number> is the line in the current program where renumbering is to begin. The default is the first line of the program. <increment> is the increment to be used in the new sequence. The default is 10.

RENUM also changes all line number references following GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB, and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, the error message "Undefined line number in xxxxx" is

printed. The incorrect line number reference is not changed by RENUM, but line number yyyyy may be changed.

Note

RENUM cannot be used to change the order of program lines (for example, RENUM 15,30 when the program has three lines numbered 10, 20, and 30) or to create line numbers greater than 65529. An "Illegal function call" error will result.

Examples

RENUM	Renumbers the entire program. The first new line number will be 10. Lines will be numbered in increments of 10.
RENUM 300,,50	Renumbers the entire program. The first new line number will be 300. Lines will be numbered in increments of 50.
RENUM 1000,900,20	Renumbers the lines from 900 up so they start with line number 1000 and are numbered in increments of 20.

2.57 RESTORE

Syntax

RESTORE [<line number>]

Purpose

To allow DATA statements to be reread from a specified line.

Remarks

After a RESTORE statement is executed, the next READ statement accesses the first item in the first DATA statement in the program. If <line number> is specified, the next READ statement accesses the first item in the specified DATA statement.

Example

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57, 68, 79
.
.
.
```

2.58 RESUME

Syntax

RESUME

RESUME 0

RESUME NEXT

RESUME <line number>

Purpose

To continue program execution after an error recovery procedure has been performed.

Remarks

Any one of the four syntax shown above may be used, depending upon where execution is to resume:

RESUME or RESUME 0	Execution resumes at the statement which caused the error.
RESUME NEXT	Execution resumes at the statement immediately following the one which caused the error.
RESUME <line number>	Execution resumes at <line number>.

A RESUME statement that is not in an error handling routine causes a "RESUME without error" message to be printed.

Example

```
10 ON ERROR GOTO 900
 .
 .
 .
900 IF (ERR = 230)AND(ERL = 90) THEN PRINT
"TRY AGAIN":RESUME 80
 .
 .
 .
```

2.59 RUN

Syntax 1 RUN [<line number>]

Purpose To execute the program currently in memory.

Remarks If <line number> is specified, execution begins on that line. Otherwise, execution begins at the lowest line number. Microsoft BASIC always returns to command level after a RUN is executed.

Example

```
RUN
```

Syntax 2 RUN <filename>[,R]

Purpose To load a file from disk into memory and run it.

Remarks <filename> is the name used when the file was SAVEd. (Your operating system may append a default filename extension if one was not supplied in the SAVE command. Refer to "Disk File Handling," in the *Microsoft BASIC User's Guide* for information about possible filename extensions under your operating system.)

RUN closes all open files and deletes the current contents of memory before loading the designated program. However, with the R option, all data files remain OPEN.

Example

```
RUN "NEWFIL",R
```

See also "Microsoft BASIC Disk I/O," in the *Microsoft BASIC User's Guide.*

Note Microsoft BASIC Compiler supports the RUN and RUN <line number> forms of the RUN statement. Microsoft BASIC Compiler does not support the R option with RUN. If you want this feature, the CHAIN statement should be used.

2.60 SAVE

Syntax SAVE <filename>[{,A|,P}]

Purpose To save a program file on disk.

Remarks <filename> is a quoted string that conforms to your operating system's requirements for filenames. (Your operating system may append a default filename extension if one was not supplied in the SAVE command. Refer to "Disk File Handling," in the *Microsoft BASIC User's Guide* for information about possible filename extensions under your operating system.) If <filename> already exists, the file will be written over.

Use the A option to save the file in ASCII format. Otherwise, Microsoft BASIC saves the file in a compressed binary format. ASCII format takes more space on the disk, but some disk access requires that files be in ASCII format. For instance, the MERGE command requires an ASCII format file, and some operating system commands such as LIST may require an ASCII format file.

Use the P option to protect the file by saving it in an encoded binary format. When a protected file is later RUN (or LOADed), any attempt to list or edit it will fail.

Examples

```
SAVE "COM2",A
SAVE "PROG",P
```

See also "Disk File Handling," in the *Microsoft BASIC User's Guide.*

2.61 STOP

Syntax STOP

Purpose To terminate program execution and return to command level.

Remarks STOP statements may be used anywhere in a program to terminate execution. When a STOP is encountered, the following message is printed:

```
Break in line nnnnn
```

Unlike the END statement, the STOP statement does not close files.

Microsoft BASIC always returns to command level after a STOP is executed. Execution is resumed by issuing a CONT command (see Section 2.8).

Example

```
10 INPUT A,B,C
20 K = A^2*5.3:L = B^3/.26
30 STOP
40 M = C*K + 100:PRINT M
RUN
? 1,2,3
BREAK IN 30
Ok
PRINT L
  30.7692
Ok
CONT
  115.9
Ok
```

2.62 SWAP

Syntax SWAP <variable>,<variable>

Purpose To exchange the values of two variables.

Remarks Any type variable may be SWAPped (integer, single precision, double precision, string), but the two variables must be of the same type or a "Type mismatch" error results.

Example

```
LIST
10 A$ = " ONE " : B$ = " ALL " : C$ = "FOR"
20 PRINT A$ C$ B$
30 SWAP A$, B$
40 PRINT A$ C$ B$
RUN
Ok
  ONE FOR ALL
  ALL FOR ONE
Ok
```

2.63 TRON/TROFF

Syntax TRON

TROFF

Purpose To trace the execution of program statements.

Remarks As an aid in debugging, the TRON statement (executed in either direct or indirect mode) enables a trace flag that prints each line number of the program as it is executed. The numbers appear enclosed in square brackets. The trace flag is disabled with the TROFF statement (or when a NEW command is executed).

Example

```
TRON
Ok
LIST
10 K = 10
20 FOR J = 1 TO 2
30 L = K + 10
40 PRINT J;K;L
50 K = K + 10
60 NEXT
70 END
Ok
RUN
[10][20][30][40] 1 10 20
[50][60][30][40] 2 20 30
[50][60][70]
Ok
TROFF
Ok
```

2.64 WAIT

Syntax

WAIT <port number>,I[,J]

where I and J are integer expressions.

Purpose

To suspend program execution while monitoring the status of a machine input port.

Remarks

The WAIT statement causes execution to be suspended until a specified machine input port develops a specified bit pattern. The data read at the port is exclusive OR'ed with the integer expression J, and then AND'ed with I. If the result is zero, Microsoft BASIC loops back and reads the data at the port again. If the result is nonzero, execution continues with the next statement. If J is omitted, it is assumed to be zero.

Important

It is possible to enter an infinite loop with the WAIT statement, in which case it will be necessary to manually restart the machine. To avoid this, WAIT must have the specified value at

<port number> during some point in the program execution.

Example

```
100 WAIT 32,2
```

2.65 WHILE...WEND

Syntax

```
WHILE <expression>
.
.
.
[<loop statements>]
.
.
WEND
```

Purpose

To execute a series of statements in a loop as long as a given condition is true.

Remarks

If <expression> is not zero (i.e., true), <loop statements> are executed until the WEND statement is encountered. Microsoft BASIC then returns to the WHILE statement and checks <expression>. If it is still true, the process is repeated. If it is not true, execution resumes with the statement following the WEND statement.

WHILE/WEND loops may be nested to any level. Each WEND will match the most recent WHILE. An unmatched WHILE statement causes a "WHILE without WEND" error, and an unmatched WEND statement causes a "WEND without WHILE" error.

Example

```
90 'BUBBLE SORT ARRAY A$
100 FLIPS = 1 'FORCE ONE PASS THRU LOOP
110 WHILE FLIPS
115         FLIPS = 0
120         FOR I = 1 TO J-1
130             IF A$(I)>A$(I + 1) THEN
                        SWAP A$(I),A$(I + 1):FLIPS = 1
140         NEXT I
150 WEND
```

2.66 WIDTH

Syntax

WIDTH [LPRINT]<integer expression>

Purpose

To set the printed line width in number of characters for the terminal or line printer.

Remarks

If the LPRINT option is omitted, the line width is set at the terminal. If LPRINT is included, the line width is set at the line printer.

<integer expression> must have a value in the range 15 to 255. The default width is 72 characters.

If <integer expression> is 255, the line width is "infinite"; that is, Microsoft BASIC never inserts a carriage return. However, the position of the cursor or the print head, as given by the POS or LPOS function, returns to zero after position 255.

Example

```
10 PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
WIDTH 18
Ok
RUN
ABCDEFGHIJKLMNOPQR
STUVWXYZ
Ok
```

2.67 WRITE

Syntax WRITE [<list of expressions>]

Purpose To output data at the terminal.

Remarks If <list of expressions> is omitted, a blank line is output. If <list of expressions> is included, the values of the expressions are output at the terminal. The expressions in the list may be numeric and/or string expressions. They must be separated by commas.

When the printed items are output, each item is separated from the last by a comma. Printed strings are delimited by quotation marks. After the last item in the list is printed, Microsoft BASIC inserts a carriage return/line feed.

WRITE outputs numeric values using the same syntax as the PRINT statement. (See Section 2.49.)

Example

```
10 A = 80:B = 90:C$ = "THAT'S ALL"
20 WRITE A,B,C$
RUN
  80, 90,"THAT'S ALL"
Ok
```

2.68 WRITE#

Syntax WRITE#<file number>,<list of expressions>

Purpose To write data to a sequential file.

Remarks <file number> is the number under which the file was OPENed in "O" mode (see "OPEN," Section 2.45). The expressions in the list are string or numeric expressions. They must be separated by commas.

The difference between WRITE# and PRINT# is that WRITE# inserts commas between the items as they are written to disk and delimits strings with quotation marks. Therefore, it is not necessary for the user to put explicit delimiters in the list. A carriage return/line feed sequence is inserted after the last item in the list is written to disk.

Example Let A$ = "CAMERA" and B$ = "93604-1"

The statement

```
WRITE#1,A$,B$
```

writes the following image to disk:

```
"CAMERA","93604-1"
```

A subsequent INPUT# statement, such as

```
INPUT#1,A$,B$
```

would input "CAMERA" to A$ and "93604-1" to B$.

Chapter 3

Microsoft BASIC Functions

Microsoft BASIC intrinsic functions are described in this chapter. The functions may be called from any program without further definition.

Arguments to functions are always enclosed in parentheses. In the syntaxes given for the functions in this chapter, the arguments have been abbreviated as follows:

X and Y	Represent any numeric expressions.
I and J	Represent integer expressions.
X$ and Y$	Represent string expressions.

If a floating-point value is supplied where an integer is required, Microsoft BASIC will round the fractional portion and use the resulting integer.

Note

With Microsoft BASIC Interpreter, only integer and single precision results are returned by functions. Double precision functions are supported only by the Microsoft BASIC Compiler.

3.1 ABS

Syntax ABS(X)

Action Returns the absolute value of the expression X.

Example

```
PRINT ABS(7*(-5))
 35
Ok
```

3.2 ASC

Syntax ASC(X$)

Action Returns a numerical value that is the ASCII code for the first character of the string X$. (See Appendix C for ASCII codes.) If X$ is null, an "Illegal function call" error is returned.

Example

```
10 X$ = "TEST"
20 PRINT ASC(X$)
RUN
 84
Ok
```

See the CHR$ function, Section 3.5, for details on ASCII-to-string conversion.

3.3 ATN

Syntax ATN(X)

Action Returns the arctangent of X in radians. Result is in the range -pi/2 to pi/2. The expression X may be any numeric type, but the evaluation of ATN is always performed in single precision.

Example

```
10 INPUT X
20 PRINT ATN(X)
RUN
? 3
 1.24905
Ok
```

3.4 CDBL

Syntax CDBL(X)

Action Converts X to a double precision number.

Example

```
10 A = 454.67
20 PRINT A;CDBL(A)
RUN
 454.67 454.6700134277344
Ok
```

3.5 CHR$

Syntax CHR$(I)

Action Returns a string whose one character is ASCII character I. (ASCII codes are listed in Appendix C.) CHR$ is commonly used to send a special character to the terminal. For instance, the BEL character (CHR$(7)) could be sent as a preface to an error message, or a form feed (CHR$(12)) could be sent to clear a terminal screen and return the cursor to the home position.

Example

```
PRINT CHR$(66)
B
Ok
```

See the ASC function, Section 3.2, for details on ASCII-to-numeric conversion.

3.6 CINT

Syntax CINT(X)

Action Converts X to an integer by rounding the fractional portion. If X is not in the range -32768 to 32767, an "Overflow" error occurs.

Example

```
PRINT CINT(45.67)
 46
Ok
```

See the CDBL and CSNG functions for details on converting numbers to the double precision and single precision data type, respectively. See also the FIX and INT functions, both of which return integers.

3.7 COS

Syntax COS(X)

Action Returns the cosine of X in radians. The calculation of COS(X) is performed in single precision.

Example

```
10 X=2*COS(.4)
20 PRINT X
RUN
 1.84212
Ok
```

See also "MKI$, MKS$, MKD$," Section 3.26 and "Disk File Handling," in the *Microsoft BASIC User's Guide.*

3.10 EOF

Syntax EOF(<file number>)

Action Returns -1 (true) if the end of a sequential file has been reached. Use EOF to test for end-of-file while INPUTting, to avoid "Input past end" errors.

Example

```
10 OPEN "I",1,"DATA"
20 C=0
30 IF EOF(1) THEN 100
40 INPUT #1,M(C)
50 C=C+1:GOTO 30
 .
 .
 .
```

3.11 EXP

Syntax EXP(X)

Action Returns e (base of natural logarithms) to the power of X. X must be <=87.3365. If EXP overflows, the "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result, and execution continues.

Example

```
10 X=5
20 PRINT EXP(X-1)
RUN
 54.5982
Ok
```

3.12 FIX

Syntax FIX(X)

Action Returns the truncated integer part of X. FIX(X) is equivalent to SGN(X)*INT(ABS(X)). The major difference between FIX and INT is that FIX does not return the next lower number for negative X.

Examples

```
PRINT FIX(58.75)
 58
Ok

PRINT FIX(-58.75)
-58
Ok
```

3.13 FRE

Syntax

```
FRE(0)
FRE(" ")
```

Action Arguments to FRE are dummy arguments. FRE returns the number of bytes in memory not being used by Microsoft BASIC.

FRE(" ") forces a garbage collection before returning the number of free bytes. Be patient: garbage collection may take 1 to 1½ minutes.

Microsoft BASIC will not initiate garbage collection until all free memory has been used up. Therefore, using FRE(" ") periodically will result in shorter delays for each garbage collection.

Example

```
PRINT FRE(0)
 14542
Ok
```

3.14 HEX$

Syntax HEX$(X)

Action Returns a string which represents the hexadecimal value of the decimal argument. X is rounded to an integer before HEX$(X) is evaluated.

Example

```
10 INPUT X
20 A$ = HEX$(X)
30 PRINT X "DECIMAL IS" A$ "HEXADECIMAL"
RUN
? 32
 32 DECIMAL IS 20 HEXADECIMAL
Ok
```

See the OCT$ function, Section 3.27, for details on octal conversion.

3.15 INKEY$

Syntax INKEY$

Action Returns either a one-character string containing a character read from the terminal or a null string if no character is pending at the terminal. No characters will be echoed. All characters are passed through to the program except for Control-C, which terminates the program. (With Microsoft BASIC Compiler, Control-C is also passed through to the program.)

Example

```
1000 'TIMED INPUT SUBROUTINE
1010 RESPONSE$ = " "
1020 FOR I% = 1 TO TIMELIMIT%
1030 A$ = INKEY$ : IF LEN(A$) = 0 THEN 1060
1040 IF ASC(A$) = 13 THEN TIMEOUT% = 0 : RETURN
1050 RESPONSE$ = RESPONSE$ + A$
1060 NEXT I%
1070 TIMEOUT% = 1 : RETURN
```

3.16 INP

Syntax INP(I)

Action Returns the byte read from port I. I must be in the range 0 to 255. INP is the complementary function to the OUT statement, Section 2.47.

Example

```
100 A = INP(255)
```

3.17 INPUT$

Syntax INPUT$(X[,[#]Y])

Action Returns a string of X characters, read from the terminal or from file number Y. If the terminal is used for input, no characters will be echoed. All control characters are passed through except Control-C, which is used to interrupt the execution of the INPUT$ function.

Example 1

```
5 'LIST THE CONTENTS OF A SEQUENTIAL FILE
IN HEXADECIMAL
10 OPEN"I",1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Example 2

```
.
.
.
100 PRINT "TYPE P TO PROCEED OR S TO STOP"
110 X$ = INPUT$(1)
120 IF X$ = "P" THEN 500
130 IF X$ = "S" THEN 700 ELSE 100
.
.
.
```

3.20 LEFT$

Syntax LEFT$(X$,I)

Action Returns a string comprised of the leftmost I characters of X$. I must be in the range 0 to 255. If I is greater than the number of characters in X$ (LEN(X$)), the entire string (X$) will be returned. If I=0, the null string (length zero) is returned.

Example

```
10 A$ = "BASIC"
20 B$ = LEFT$(A$,5)
30 PRINT B$
BASIC
Ok
```

Also see the MID$ and RIGHT$ functions, Sections 3.25 and 3.30, respectively.

3.21 LEN

Syntax LEN(X$)

Action Returns the number of characters in X$. Nonprinting characters and blanks are counted.

Example

```
10 X$ = "PORTLAND, OREGON"
20 PRINT LEN(X$)
 16
Ok
```

3.22 LOC

Syntax LOC(<file number>)

where <file number> is the number under which the file was OPENed.

Action With random disk files, LOC returns the record number just read or written from a GET or PUT statement. If the file was opened but no disk I/O has been performed yet, LOC returns a 0. With sequential files, LOC returns the number of sectors (128-byte blocks) read from or written to the file since it was OPENed.

Example

```
200 IF LOC(1)>50 THEN STOP
```

3.23 LOG

Syntax LOG(X)

Action Returns the natural logarithm of X. X must be greater than zero.

Example

```
PRINT LOG(45/7)
 1.86075
Ok
```

3.24 LPOS

Syntax LPOS(X)

Action Returns the current position of the line printer print head within the line printer's buffer. Does not necessarily give the physical position of the print head. X is a dummy argument.

Example

```
100 IF LPOS(X)>60 THEN LPRINT CHR$(13)
```

3.25 MID$

Syntax MID$(X$,I[,J])

Action Returns a string of length J characters from X$, beginning with the Ith character. I and J must be in the range 1 to 255. If J is omitted or if there are fewer than J characters to the right of the Ith character, all rightmost characters beginning with the Ith character are returned. If I is greater than the number of characters in X$ (LEN(X$)), MID$ returns a null string.

Example

```
LIST
10 A$ = "GOOD "
20 B$ = "MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,9,7)
Ok
RUN
GOOD EVENING
Ok
```

Also see the LEFT$ and RIGHT$ functions, Sections 3.20 and 3.30, respectively.

If I=0 is specified, the "Illegal function call" error message will be returned.

3.26 MKI$, MKS$, MKD$

Syntax

```
MKI$(<integer expression>)
MKS$(<single precision expression>)
MKD$(<double precision expression>)
```

Action Convert numeric values to string values. Any numeric value that is placed in a random file buffer with an LSET or RSET statement must be converted to a string. MKI$ converts an integer to a 2-byte string. MKS$ converts a single precision number to a 4-byte string. MKD$ converts a double precision number to an 8-byte string.

Example

```
90 AMT = (K + T)
100 FIELD #1,8 AS D$,20 AS N$
110 LSET D$ = MKS$(AMT)
120 LSET N$ = A$
130 PUT #1
.
.
```

Also see "CVI, CVS, CVD," Section 3.9, and "Disk File Handling," in the *Microsoft BASIC User's Guide.*

3.27 OCT$

Syntax

OCT$(X)

Action

Returns a string which represents the octal value of the decimal argument. X is rounded to an integer before OCT$(X) is evaluated.

Example

```
PRINT OCT$(24)
 30
Ok
```

See the HEX$ function, Section 3.14, for details on hexadecimal conversion.

3.28 PEEK

Syntax

PEEK(I)

Action

Returns the byte read from the indicated memory location (I).

Remarks

The returned value is an integer in the range 0 to 255. I must be in the range -32768 to 65535. (For the interpretation of a negative value of I, see "VARPTR," Section 3.43.)

PEEK is the complementary function of the POKE statement.

Example

```
A = PEEK(&H5A00)
```

3.29 POS

Syntax

```
POS(I)
```

Action

Returns the current cursor position. The leftmost position is 1. X is a dummy argument.

Example

```
IF POS(X)>60 THEN PRINT CHR$(13)
```

Also see the LPOS function, Section 3.24.

3.30 RIGHT$

Syntax

```
RIGHT$(X$,I)
```

Action

Returns the rightmost I characters of string X$. If I is equal to the number of characters in X$ (LEN(X$)), RIGHT$ returns X$. If I=0, the null string (length zero) is returned.

Example

```
10 A$ = "DISK BASIC"
20 PRINT RIGHT$(A$,5)
RUN
BASIC
Ok
```

Also see the LEFT$ and MID$ functions, Sections 3.20 and 3.25, respectively.

3.31 RND

Syntax RND[(X)]

Action Returns a random number between 0 and 1. The same sequence of random numbers is generated each time the program is RUN unless the random number generator is reseeded (see "RANDOMIZE," Section 2.53). However, X<0 always restarts the same sequence for any given X.

X>0 or X omitted generates the next random number in the sequence. X=0 repeats the last number generated.

Example

```
10 FOR I = 1 TO 5
20 PRINT INT(RND*100);
30 NEXT
RUN
  24 30 31 51 5
Ok
```

Note The values produced by the RND function may vary with different implementations of Microsoft BASIC.

3.32 SGN

Syntax SGN(X)

Action

If X>0, SGN(X) returns 1.
If X=0, SGN(X) returns 0.
If X<0, SGN(X) returns -1.

Example ON SGN(X)+2 GOTO 100,200,300

branches to 100 if X is negative, 200 if X is 0, and 300 if X is positive.

3.33 SIN

Syntax SIN(X)

Action Returns the sine of X in radians. SIN(X) is calculated in single precision. COS(X)=SIN(X+3.14159/2).

Example

```
PRINT SIN(1.5)
 .997495
Ok
```

Also see the COS(X) function, Section 3.7.

3.34 SPACE$

Syntax SPACE$(X)

Action Returns a string of spaces of length X. The expression X is rounded to an integer and must be in the range 0 to 255.

Example

```
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
  1
    2
      3
        4
          5
Ok
```

Also see the SPC function, Section 3.35.

3.35 SPC

Syntax SPC(I)

Action Prints I blanks on the terminal. SPC may only be used with PRINT and LPRINT statements. I must be in the range 0 to 255. A";" is assumed to follow the SPC(I) command.

Example

```
PRINT "OVER" SPC(15) "THERE"
OVER               THERE
Ok
```

Also see the SPACE$ function, Section 3.34.

3.36 SQR

Syntax SQR(X)

Action Returns the square root of X. X must be >=0.

Example

```
10 FOR X = 10 TO 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT
RUN
  10          3.16228
  15          3.87298
  20          4.47214
  25          5
Ok
```

3.37 STR$

Syntax STR$(X)

Action Returns a string representation of the value of X.

Example

```
5 REM ARITHMETIC FOR KIDS
10 INPUT "TYPE A NUMBER";N
20 ON LEN(STR$(N)) GOSUB 30,100,200,300,400,500
 .
 .
 .
```

Also see the VAL function, Section 3.42.

3.38 STRING$

Syntaxes

```
STRING$(I,J)
STRING$(I,X$)
```

Action Returns a string of length I whose characters all have ASCII code J or the first character of X$.

Example

```
10 X$ = STRING$(10,45)
20 PRINT X$ "MONTHLY REPORT" X$
RUN
----------MONTHLY REPORT----------
Ok
```

3.39 TAB

Syntax TAB(I)

Action Spaces to position I on the terminal. If the current print position is already beyond space I, TAB goes to that position on the next line. Space 1 is the left-

most position, and the rightmost position is the width minus one. I must be in the range 1 to 255. TAB may only be used in PRINT and LPRINT statements.

Example

```
10 PRINT "NAME" TAB(25) "AMOUNT" : PRINT
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "G. T. JONES","$25.00"
RUN
NAME                      AMOUNT
G. T. JONES               $25.00
Ok
```

3.40 TAN

Syntax TAN(X)

Action Returns the tangent of X in radians. TAN(X) is calculated in single precision. If TAN overflows, the "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result, and execution continues.

Example

```
10 Y = Q*TAN(X)/2
```

3.41 USR

Syntax USR[<digit>](X)

Action Calls the user's assembly language subroutine with the argument X. <digit> is in the range 0 to 9 and corresponds to the digit supplied with the

DEF USR statement for that routine. If <digit> is omitted, USR0 is assumed. See "Assembly Language Subroutines," in the *Microsoft BASIC User's Guide.*

Example

```
40 B = T*SIN(Y)
50 C = USR(B/2)
60 D = USR(B/3)
 .
 .
 .
```

3.42 VAL

Syntax

VAL(X$)

Action

Returns the numerical value of string X$. The VAL function also strips leading blanks, tabs, and linefeeds from the argument string. For example,

VAL(" -3")

returns -3.

Example

```
10 READ NAME$,CITY$,STATE$,ZIP$
20 IF VAL(ZIP$)<90000 OR VAL(ZIP$)>96699 THEN
PRINT NAME$ TAB(25) "OUT OF STATE"
30 IF VAL(ZIP$)>=90801 AND VAL(ZIP$)<=90815
THEN PRINT NAME$ TAB(25) "LONG BEACH"
 .
 .
 .
```

See the STR$ function, Section 3.37, for details on numeric-to-string conversion.

3.43 VARPTR

Syntax 1 VARPTR(<variable name>)

Syntax 2 VARPTR(#<file number>)

Action *Syntax 1*

Returns the address of the first byte of data identified with <variable name>. A value must be assigned to <variable name> prior to execution of VARPTR. Otherwise an "Illegal function call" error results. Any type variable name may be used (numeric, string, array). For string variables, the address of the first byte of the string descriptor is returned (see "BASIC Assembly Language Subroutines," in the *Microsoft BASIC User's Guide* for discussion of the string descriptor). The address returned will be an integer in the range -32768 to 32767. If a negative address is returned, add it to 65536 to obtain the actual address.

VARPTR is usually used to obtain the address of a variable or array so it may be passed to an assembly language subroutine. A function call of the form VARPTR(A(0)) is usually specified when passing an array, so that the lowest-addressed element of the array is returned.

Note All simple variables should be assigned before calling VARPTR for an array, because the addresses of the arrays change whenever a new simple variable is assigned.

Syntax 2

For sequential files, returns the starting address of the disk I/O buffer assigned to <file number>. For random files, returns the address of the FIELD buffer assigned to <file number>.

Example

```
100 X = USR(VARPTR(Y))
```

Appendices

Appendix A

Error Codes and Error Messages

Code	Number	Message
NF	1	NEXT without FOR A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR statement variable.
SN	2	Syntax error A line is encountered that contains some incorrect sequence of characters (such as unmatched parenthesis, misspelled command or statement, incorrect punctuation, etc.). Microsoft BASIC automatically enters edit mode at the line that caused the error.
RG	3	Return without GOSUB A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.
OD	4	Out of data A READ statement is executed when there are no DATA statements with unread data remaining in the program.
FC	5	Illegal function call A parameter that is out of range is passed to a math or string function. An FC error may also occur as the result of: 1. A negative or unreasonably large subscript.

2. A negative or zero argument with LOG.
3. A negative argument to SQR.
4. A negative mantissa with a noninteger exponent.
5. A call to a USR function for which the starting address has not yet been given.
6. An improper argument to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, or ON...GOTO.

OV 6 Overflow

The result of a calculation is too large to be represented in Microsoft BASIC number format. If underflow occurs, the result is zero and execution continues without an error.

OM 7 Out of memory

A program is too large, or has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated.

UL 8 Undefined line

A nonexistent line is referenced in a GOTO, GOSUB, IF...THEN...ELSE, or DELETE statement.

BS 9 Subscript out of range

An array element is referenced either with a subscript that is outside the dimensions of the array or with the wrong number of subscripts.

DD 10 Redimensioned array

Two DIM statements are given for the same array; or, a DIM statement is given for an array after the default dimension of 10 has been established for that array.

/0	11	Division by zero A division by zero is encountered in an expression; or, the operation of involution results in zero being raised to a negative power. Machine infinity with the sign of the numerator is supplied as the result of the division, or positive machine infinity is supplied as the result of the involution, and execution continues.
ID	12	Illegal direct A statement that is illegal in direct mode is entered as a direct mode command.
TM	13	Type mismatch A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.
OS	14	Out of string space String variables have caused BASIC to exceed the amount of free memory remaining. Microsoft BASIC will allocate string space dynamically, until it runs out of memory.
LS	15	String too long An attempt is made to create a string more than 255 characters long.
ST	16	String formula too complex A string expression is too long or too complex. The expression should be broken into smaller expressions.
CN	17	Can't continue An attempt is made to continue a program that: 1. Has halted due to an error.

2. Has been modified during a break in execution.

3. Does not exist.

UF 18 Undefined user function

A USR function is called before the function definition (DEF statement) is given.

19 No RESUME

An error handling routine is entered but contains no RESUME statement.

20 RESUME without error

A RESUME statement is encountered before an error handling routine is entered.

21 Unprintable error

An error message is not available for the error condition which exists.

22 Missing operand

An expression contains an operator with no operand following it.

23 Line buffer overflow

An attempt has been made to input a line that has too many characters.

26 FOR without NEXT

A FOR statement was encountered without a matching NEXT.

29 WHILE without WEND

A WHILE statement does not have a matching WEND.

30 WEND without WHILE

A WEND statement was encountered without a matching WHILE.

Disk Errors

Number	Message
50	Field overflow A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.
51	Internal error An internal malfunction has occurred in Microsoft BASIC. Report to Microsoft the conditions under which the message appeared.
52	Bad file number A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.
53	File not found A LOAD, KILL, or OPEN statement references a file that does not exist on the current disk.
54	Bad file mode An attempt is made to use PUT, GET, or LOF with a sequential file, to LOAD a random file, or to execute an OPEN statement with a file mode other than I, O, or R.
55	File already open A sequential output mode OPEN statement is issued for a file that is already open; or a KILL statement is given for a file that is open.

57 Disk I/O error

An I/O error occurred on a disk I/O operation. It is a fatal error; i.e., the operating system cannot recover from the error.

58 File already exists

The filename specified in a NAME statement is identical to a filename already in use on the disk.

61 Disk full

All disk storage space is in use.

62 Input past end

An INPUT statement is executed after all the data in the file has been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end-of-file.

63 Bad record number

In a PUT or GET statement, the record number is either greater than the maximum allowed (32,767) or equal to zero.

64 Bad file name

An illegal form is used for the filename with a LOAD, SAVE, KILL, or OPEN statement (e.g., a filename with too many characters).

66 Direct statement in file

A direct statement is encountered while LOADing an ASCII-format file. The LOAD is terminated.

67 Too many files

An attempt is made to create a new file (using SAVE or OPEN) when all 255 directory entries are full.

Appendix B
Mathematical Functions

Derived Functions

Functions that are not intrinsic to Microsoft BASIC may be calculated as follows.

Function	Microsoft BASIC Equivalent
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X)=-ATN(X/SQR(-X*X+1)) +1.5708
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X-1)) +SGN(SGN(X)-1)*1.5708
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X-1)) +(SGN(X)-1)*1.5708
INVERSE COTANGENT	ARCCOT(X)=ATN(X)+1.5708
HYPERBOLIC SINE	SINH(X)=(EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X)=(EXP(X)-EXP(-X))/ (EXP(X)+EXP(-X))
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X)=(EXP(X)+EXP(-X))/ (EXP(X)-EXP(-X))

INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X-1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X)=LOG((X+1)/(X-1))/2

Appendix C
ASCII Character Codes

Dec	Hex	CHR	Dec	Hex	CHR	Dec	Hex	CHR
000	00H	NUL	043	2BH	+	086	56H	V
001	01H	SOH	044	2CH	,	087	57H	W
002	02H	STX	045	2DH	-	088	58H	X
003	03H	ETX	046	2EH	.	089	59H	Y
004	04H	EOT	047	2FH	/	090	5AH	Z
005	05H	ENQ	048	30H	0	091	5BH	[
006	06H	ACK	049	31H	1	092	5CH	\
007	07H	BEL	050	32H	2	093	5DH	]
008	08H	BS	051	33H	3	094	5EH	^
009	09H	HT	052	34H	4	095	5FH	_
010	0AH	LF	053	35H	5	096	60H	'
011	0BH	VT	054	36H	6	097	61H	a
012	0CH	FF	055	37H	7	098	62H	b
013	0DH	CR	056	38H	8	099	63H	c
014	0EH	SO	057	39H	9	100	64H	d
015	0FH	SI	058	3AH	:	101	65H	e
016	10H	DLE	059	3BH	;	102	66H	f
017	11H	DC1	060	3CH	<	103	67H	g
018	12H	DC2	061	3DH	=	104	68H	h
019	13H	DC3	062	3EH	>	105	69H	i
020	14H	DC4	063	3FH	?	106	6AH	j
021	15H	NAK	064	40H	@	107	6BH	k
022	16H	SYN	065	41H	A	108	6CH	l
023	17H	ETB	066	42H	B	109	6DH	m
024	18H	CAN	067	43H	C	110	6EH	n
025	19H	EM	068	44H	D	111	6FH	o
026	1AH	SUB	069	45H	E	112	70H	p
027	1BH	ESCAPE	070	46H	F	113	71H	q
028	1CH	FS	071	47H	G	114	72H	r
029	1DH	GS	072	48H	H	115	73H	s
030	1EH	RS	073	49H	I	116	74H	t
031	1FH	US	074	4AH	J	117	75H	u
032	20H	SPACE	075	4BH	K	118	76H	v
033	21H	!	076	4CH	L	119	77H	w
034	22H	"	077	4DH	M	120	78H	x
035	23H	#	078	4EH	N	121	79H	y
036	24H	$	079	4FH	O	122	7AH	z
037	25H	%	080	50H	P	123	7BH	{
038	26H	&	081	51H	Q	124	7CH	|
039	27H	'	082	52H	R	125	7DH	}
040	28H	(	083	53H	S	126	7EH	~
041	29H	)	084	54H	T	127	7FH	DEL
042	2AH	*	085	55H	U			

Dec=decimal, Hex=hexadecimal (H), CHR=character, LF=Line Feed, FF=Form Feed, CR=Carriage Return, DEL=Rubout

Appendix D

Microsoft BASIC Reserved Words

The following is a list of reserved words used in Microsoft BASIC.

ABS
AND
ASC
ATN
AUTO
CALL
CDBL
CHAIN
CHR$
CINT
CLEAR
CLOSE
COMMON
CONT
COS
CSAVE
CSNG
CVD
CVI
CVS
DATA
DEFDBL
DEFINT
DEFSNG
DEFSTR
DEF USR
DELETE
DIM
EDIT
ELSE
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP
FIELD
FILES
FIX
FN
FOR
FRE
GET
GOSUB
HEX$
IF
IMP
INKEY$
INP
INPUT
INPUT#
INPUT$
INSTR
INT
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOF
LOG
LPOS
LPRINT
LSET
MERGE
MID$
MKD$
MKI$
MKS$
MOD
NAME
NEW
NEXT
NOT
OCT$
ON
OPEN
OPTION
OR
OUT
PEEK
POKE
POS
PRINT
PUT
RANDOMIZE
READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SGN
SIN
SPACE$
SPC
SQR
STEP
STOP
STR$
STRING$
SWAP
SYSTEM
TAB
TAN
THEN
TO
TROFF
TRON
USING
USR
VAL
VARPTR
WAIT
WEND
WHILE
WIDTH
WRITE
WRITE#
XOR

98074

Microsoft® BASIC

Reference Manual

PART 2

Microsoft Corporation

Index