MICROSOFT Corporation

QBASIC Command Reference

for

QBASIC V1.1

Copyright and Trademarks

Contents

Index of Commands and Functions

Introduction

Contents of Introduction

This manual is based on the original QBASIC V1.1 online help system.

Syntax Conventions

KEYWORDS	Items in capital letters are Basic keywords. Keywords are a required part of the statement syntax, unless they are enclosed in brackets.
placeholders	Items in lowercase are placeholders for information you must supply in the statement (such as a filename$). The QBasic syntax uses data-type suffixes for placeholders that must be a specific data type. Placeholders that can be more than one data type do not have data-type suffixes.
[optional item]	Items inside square brackets are optional.
{choice1 | choice2}	Braces and a vertical bar indicate a choice between two or more items. You must use one of the items in the statement unless the braces are enclosed in square brackets.
item, item, ...	A horizontal three-dot ellipsis means more of the preceding items can be used in a single-line statement.
Beginning keyword . . . Ending keyword	A vertical three-dot ellipsis is used to describe multiline statements (or block-structured statements). It means that other statements can be used between the beginning and the end of the block.

Character Set in QBASIC

The Microsoft Basic character set includes alphabetic characters (A-Z, a-z), numeric characters (0-9 and A-F or a-f for hexadecimal numbers), and special characters. Some characters have special meanings in Basic:

Data-Type Suffixes			
!	Single-precision	%	Integer
#	Double-precision	&	Long-integer
$	String		

Mathematical Operators	
*	Multiplication symbol
-	Minus sign
/	Division symbol (slash)
=	Relational operator or assignment symbol
>	Greater than
+	Plus sign
.	Decimal point
<	Less than
\	Integer division symbol (backslash)
^	Exponentiation symbol (up arrow or caret)

Special	
'	Comment line (single quote)
;	Controls PRINT and INPUT statement output
,	Controls PRINT and INPUT statement output
:	Separates multiple statements on a single line
?	INPUT statement prompt
_	Line continuation underscore (reserved for compatibility with other versions of Basic but not supported by QBasic)

Keywords by Programming Task

Programming task	Keywords included in this list
Control program flow	DO...LOOP, END, EXIT, FOR...NEXT, IF...THEN...ELSE, GOSUB...RETURN, GOTO, ON...GOSUB, ON...GOTO, SELECT CASE, STOP, SYSTEM
Declare constants and variables	CONST, DATA, DIM, ERASE, OPTION BASE, READ, REDIM, REM, RESTORE, SWAP, TYPE...END TYPE
Define and call Basic procedures	CALL, DECLARE, EXIT, FUNCTION, RUN, SHELL, SHARED, STATIC, SUB
Device input/output	CLS, CSRLIN, INKEY$, INP, INPUT, KEY (Assignment), LINE INPUT, LOCATE, LPOS, LPRINT, LPRINT USING, OPEN COM, OUT, POS, PRINT, PRINT USING, SPC, SCREEN Function, TAB, VIEW PRINT, WAIT, WIDTH
Display graphic images	CIRCLE, COLOR, GET (Graphics), LINE, PAINT, PALETTE, PCOPY, PMAP, POINT, PRESET, PSET, PUT (Graphics), SCREEN Statement, VIEW, WINDOW
DOS file system commands	CHDIR, KILL, MKDIR, NAME, RMDIR
File input/output	CLOSE, EOF, FILEATTR, FREEFILE GET (File I/O), INPUT, INPUT$, LINE INPUT, LOC, LOCK, LOF, OPEN, PUT (File I/O), SEEK Function, SEEK Statement, UNLOCK, WRITE
Manage memory	CLEAR, FRE, PEEK, POKE
Manipulate strings	ASC, CHR$, HEX$, INSTR, LCASE$, LEFT$, LEN, LSET, LTRIM$, MID$ Function, MID$ Statement, OCT$, RIGHT$, RSET, RTRIM$ SPACE$, STR$, STRING$, UCASE$, VAL
Perform mathematical calculations	ABS, ASC, ATN, CDBL, CINT, CLNG, COS, CSNG, CVDMBF, CVSMBF, EXP, INT, LOG, RANDOMIZE, RND, SGN, SIN, SQR, TAN, TIME$ Function
Set traps for events and errors	COM, ERDEV, ERDEV$, ERL, ERR, ERROR, KEY (Event Trapping), ON COM, ON ERROR, ON KEY, ON PEN, ON PLAY, ON STRIG, ON TIMER, PEN, PLAY (Event Trapping), RESUME, RETURN, STRIG, TIMER Function, TIMER Statement

Environment Limits

This section lists the limits for the following items in the QBasic environment:

Code and Data Limits

	Maximum
Total Environment (Sum of code and data in memory at any time)	160k characters

Name, String, and Number Limits

	Maximum	Minimum
Variable-name length	40 characters	1 character
String length	32,767 characters	0 characters
Integers	32,767	-32,768
Long integers	2,147,483,647	-2,147,483,648
Single-precision numbers:		
Positive	3.402823E+38	2.802597E-45
Negative	-2.802597E-45	-3.402823E+38
Double-precision numbers:		
Positive	1.79769313486231D+308	4.940656458412465D-324
Negative	-4.940656458412465D-324	-1.79769313486231D+308

Array Limits

	Maximum	Minimum
Array size (all elements):		
Static	65,535 bytes (64K)	1 byte
Dynamic	65,535 bytes (64K)	
Number of dimensions allowed	60	1
Dimensions allowed if unspecified	8	1
Array subscript value	32,767	-32,768

Note: The maximum range between array subscript values is 32,767.

Procedure and File Limits

	Maximum	Minimum
Procedure size	64K	0
Number of arguments passed	60	0
Data file numbers	255	1
Data file record number	2,147,483,647	1
Data file record size	32K	1 byte
Data file size	Available disk space	0
Path names	127 characters	1 character
Error message numbers	255	1

QBASIC Command Line Options

These options can be typed on the DOS command line following the QBasic command:

```
QBASIC [/B] [/EDITOR] [/G] [/H] [/MBF] [/NOHI] [[/RUN] sourcefile]
```

Option	Description
/B	Allows the use of a composite (monochrome) monitor with a color graphics card. The /B option displays QBasic in monochrome if you have a color monitor.
/EDITOR	Invokes the MS-DOS Editor text editor. Can be abbreviated as /ED.
/G	Sets QBasic to update a CGA screen as fast as possible (works only with machines using CGA monitors). If you see snow (dots flickering on the screen) when QBasic updates your screen, your hardware cannot fully support this option. If you prefer a clean screen, restart QBasic without the /G option.
/H	Displays the maximum number of lines possible on your hardware.
/MBF	Causes the QBasic conversion functions (CVS, CVD, MKS$, MKD$) to treat IEEE-format numbers as Microsoft-Binary-format numbers.
/NOHI	Allows the use of a monitor that does not support high intensity. Not for use with Compaq laptop computers.
sourcefile	Names the file to load when QBasic starts. To load a file created with GW-BASIC or BASICA, the file must be saved from GW-BASIC or BASICA with the ,A option.
/RUN sourcefile	Causes QBasic to load and run a program file before displaying it.

1 Data, Constants and Variables

Content of section 1

$STATIC
$DYNAMIC

Set the default array storage.

```
{REM | '} $STATIC
{REM | '} $DYNAMIC
```

- {REM | '} REM or a remark character (') must precede metacommands.
- $STATIC Specifies that arrays declared in subsequent DIM statements are static arrays (unless they are declared in a non-static SUB or FUNCTION procedure). Array storage is allocated when you start the program, and remains fixed.
- $DYNAMIC Specifies that arrays declared in subsequent DIM statements are dynamic arrays. Array storage is allocated dynamically while the program runs.

- DIM and REDIM usually provide a better way to specify whether arrays are dynamic or static.

See Also DIM, REDIM REM SHARED, STATIC

COMMON

Defines global variables that can be shared throughout a program or between chained programs.

```
COMMON [SHARED] variablelist
```

- _ SHARED — Indicates that variables are shared with all SUB or FUNCTION procedures.
- _ variablelist — One or more variables to be shared:

```
variable[( )] [AS type] [, variable[( )] [AS type]]...
```

  - variable — A Basic variable name. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
  - type — The data type of the variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type).

- _ Unless it has been declared as a static array in a preceding DIM statement, an array variable in a COMMON statement is a dynamic array. Its dimensions must be set in a later DIM or REDIM statement.

See Also CHAIN, DIM, REDIM, FUNCTION, SHARED, STATIC, SUB

CONST

Declares one or more symbolic constants.

```
CONST constantname = expression [,constantname = expression]...
```

_ constantname	The name of the constant. This name can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
_ expression	An expression that is assigned to the constant. The expression can consist of literals (such as 1.0), other constants, any arithmetic or logical operators except exponentiation (^), or a single literal string.

Example:

```
CONST PI = 3.141593
INPUT "Radius of Circle: "; r
PRINT "Area = "; PI * r ^ 2
```

DATA
READ
RESTORE

DATA specifies values to be read by subsequent READ statements.
READ reads those values and assigns them to variables.
RESTORE allows READ to reread values in specified DATA statements.

```
DATA constant[,constant]...
READ variablelist
RESTORE [line]
```

_ constant	One or more numeric or string constants specifying the data to be read. String constants containing commas, colons, or leading or trailing spaces are enclosed in quotation marks (" ").
_ variablelist	One or more variables, separated by commas, that are assigned data values. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
_ line	The label or line number of a DATA statement. If line is omitted, the next READ statement reads values in the first DATA statement in the program.

_ DATA statements can be entered only at the module level. They cannot be used in procedures.

Example:

```
FOR i% = 1 TO 3
    READ a%, b$
    PRINT a%, b$
    RESTORE
NEXT i%
DATA 1, "Repeat"
```

Data Types

INTEGER LONG SINGLE DOUBLE STRING

Specify the data type for a variable in a declarative statement or parameter list:

- INTEGER — A 16-bit signed integer variable.
- LONG — A 32-bit signed integer variable.
- SINGLE — A single-precision 32-bit floating-point variable.
- DOUBLE — A double-precision 64-bit floating-point variable.
- STRING * n% — A fixed-length string variable n% bytes long.
- STRING — A variable-length string variable.

See Also AS, Basic Character Set, COMMON, DECLARE, DEF FN, DIM, REDIM, FUNCTION, SHARED, STATIC, SUB, TYPE

DEFINT
DEFLNG
DEFSNG
DEFDBL
DEFSTR

Sets the default data type for variables, DEF FN functions, and FUNCTION procedures.

```
DEFINT letterrange [,letterrange]...
DEFLNG letterrange [,letterrange]...
DEFSNG letterrange [,letterrange]...
DEFDBL letterrange [,letterrange]...
DEFSTR letterrange [,letterrange]...
```

- letterrange — A letter or range of letters (such as A-M). QBasic sets the default data type for variables, DEF FN functions, and FUNCTION procedures whose names begin with the specified letter or letters as follows:

Statement	Default Data Type
DEFINT	Integer
DEFLNG	Long integer
DEFSNG	Single precision
DEFDBL	Double precision
DEFSTR	String

- A data-type suffix (%, &, !, #, or $) always takes precedence over a DEFtype statement.
- Single-precision is the default data type if you do not specify a DEFtype statement.
- After you specify a DEFtype statement in your program, QBasic automatically inserts a corresponding DEFtype statement in each procedure you create.

Example:

```
DEFDBL A-Z
a = SQR(3)
PRINT "Square root of 3 = "; a
```

DIM
REDIM

DIM declares an array or specifies a data type for a nonarray variable.
REDIM declares or resizes a dynamic array, erasing any previous values.

```
DIM [SHARED] variable[(subscripts)] [AS type]
             [,variable[(subscripts)] [AS type]]...
REDIM [SHARED] variable(subscripts) [AS type]
               [,variable(subscripts) [AS type]]...
```

- SHARED — Specifies that variables are shared with all SUB or FUNCTION procedures in the module.
- variable — The name of an array or variable.
- subscripts — Dimensions of the array, expressed as follows:

  ```
  [lower TO] upper [,[lower TO] upper]...
  ```

  lower — The lower bound of the array's subscripts. The default lower bound is zero.
  upper — The upper bound.

- AS type — Declares the data type of the array or variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type).

- DIM declares either static or dynamic arrays. Unless array storage has been determined by $STATIC, $DYNAMIC, or COMMON, arrays dimensioned with numbers are static and arrays dimensioned with variables are dynamic. REDIM always declares dynamic arrays.
- Static array storage is allocated when you start a program and remains fixed. Dynamic array storage is allocated while a program runs.

Example:

```
' $DYNAMIC
DIM A(49, 49)
REDIM A(19, 14)
```

See Also COMMON ERASE OPTION BASE SHARED, STATIC

ERASE

Reinitializes array elements or frees dynamic array storage space.

```
ERASE arrayname [,arrayname]...
```

- arrayname The name of an array.

- For static arrays, ERASE sets each element of a numeric array to zero and each element of a string array to null.
- For dynamic arrays, ERASE frees the memory used by the array. You must redeclare the array's dimensions with REDIM or DIM before using it.

Example:

```
DIM a%(0)
a%(0) = 6
PRINT "Before: "; a%(0)
ERASE a%
PRINT "After: "; a%(0)
```

See Also CLEAR DIM, REDIM

LBOUND()
UBOUND()

Return the lower and upper bound (smallest or largest available subscript) for the specified array dimension.

```
LBOUND(array[,dimension%])
UBOUND(array[,dimension%])
```

_ array	The name of the array.
_ dimension%	Indicates the array dimension whose lower or upper bound is returned. Use 1 for the first dimension, 2 for the second dimension, etc. The default is 1.

Example:

```
DIM a%(1 TO 3, 2 TO 7)
PRINT LBOUND(a%, 1), UBOUND(a%, 2)
```

See Also DIM, REDIM

[LET]

Assigns the value of an expression to a variable.

[LET] variable=expression

- variable — Any variable. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
- expression — Any expression that provides a value to assign.

- Use of the optional LET keyword is not recommended. The variable=expression assignment statement performs the same action with or without LET.

See Also LSET, RSET

OPTION BASE

Sets the default lower bound for array subscripts.

OPTION BASE {0 | 1}

- The DIM statement TO clause provides a better way to set the lower bound of an array subscript.

See Also DIM, REDIM LBOUND, UBOUND

SWAP

Exchanges the values of two variables.

```
SWAP variable1, variable2
```

_	variable1 and variable2	Two variables of the same data type.

Example:

```
a% = 1: b% = 2
PRINT "Before: "; a%, b%
SWAP a%, b%
PRINT "After: "; a%, b%
```

TYPE

Defines a data type containing one or more elements.

```
TYPE usertype
  elementname AS typename
  [elementname AS typename]
.
.
.
END TYPE
```

_ usertype	The name of the data type being defined. The name can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
_ elementname	An element of the user-defined data type.
_ typename	The element's type (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type).

_ Use DIM, REDIM, COMMON, STATIC, or SHARED to create a variable of a user-defined data type.

Example:

```
TYPE Card
    Suit AS STRING * 9
    Value AS INTEGER
END TYPE
DIM Deck(1 TO 52) AS Card
Deck(1).Suit = "Club"
Deck(1).Value = 2
PRINT Deck(1).Suit, Deck(1).Value
```

See Also COMMON DIM, REDIM SHARED, STATIC

2 Input-Output Devices

Content of section 2

CLS

Clears the screen.

```
CLS [{0 | 1 | 2}]
```

```
    CLS      Clears either the text or graphics viewport. If a graphics
             viewport has been set (using VIEW), clears only the graphics
             viewport. Otherwise, clears the text viewport or entire screen.
    CLS 0    Clears the screen of all text and graphics.
    CLS 1    Clears the graphics viewport or the entire screen if no graphics
             viewport has been set.
    CLS 2    Clears the text viewport.
```

See Also VIEW VIEW PRINT WINDOW

LOCATE
CSRLIN
POS()

LOCATE moves the cursor to a specified position on the screen.
CSRLIN returns the current row position of the cursor.
POS returns the current column position of the cursor.

```
LOCATE  [row%] [,[column%] [,[cursor%] [,start% [,stop%]]]]
CSRLIN
POS(expression)
```

_ row% and column%	The number of the row and column to which the cursor moves.
_ cursor%	Specifies whether the cursor is visible: 0 = invisible, 1 = visible
_ start% and stop%	Integer expressions in the range 0 through 31 that specify the first and last cursor scan lines. You can change the cursor size by changing the cursor scan lines.
_ expression	Any expression.

Example:

```
    CLS
    LOCATE 5, 5
    MyRow% = CSRLIN
    MyCol% = POS(0)
    PRINT "Position 1 (Press any key)"
    DO
    LOOP WHILE INKEY$ = ""
    LOCATE (MyRow% + 2), (MyCol% + 2)
    PRINT "Position 2"
```

INKEY$

Reads a character from the keyboard.

INKEY$

- INKEY$ returns a null string if there is no character to return.
- For standard keys, INKEY$ returns a 1-byte string containing the character read.
- For extended keys, INKEY$ returns a 2-byte string made up of the null character (ASCII 0) and the keyboard scan code.

Example:

```
PRINT "Press Esc to exit..."
DO
LOOP UNTIL INKEY$ = CHR$(27)    '27 is the ASCII code for Esc.
```

See Also Keyboard Scan Codes

INP()
OUT

```
INP returns a byte read from a hardware I/O port.
OUT sends a byte to a hardware I/O port.

INP(port%)
OUT port%, data%

    _ port%     A number in the range 0 through 65,535 that identifies
                the port.
    _ data%     A numeric expression in the range 0 through 255 to send
                to the port.

Example:
    x% = INP(&H3FC)          'Read COM1 Modem Control Register.
    OUT &H3FC, (x% XOR 1)    'Change Data Terminal Ready bit.

See Also    WAIT
```

INPUT
LINE INPUT

INPUT reads input from the keyboard or a file. LINE INPUT reads a line of up to 255 characters from the keyboard or a file.

```
INPUT [;] ["prompt"{; | ,}] variablelist
LINE INPUT [;] ["prompt";] variable$
INPUT #filenumber%, variablelist
LINE INPUT #filenumber%, variable$
```

- prompt — An optional literal string that is displayed before the user enters data. A semicolon after prompt appends a question mark to the prompt string.
- variablelist — One or more variables, separated by commas, in which data entered from the keyboard or read from a file is stored. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
- variable$ — Holds a line of characters entered from the keyboard or read from a file.
- filenumber% — The number of an open file.

- INPUT uses a comma as a separator between entries. LINE INPUT reads all characters up to a carriage return.
- For keyboard input, a semicolon immediately after INPUT keeps the cursor on the same line after the user presses the Enter key.

Example:

```
CLS
OPEN "LIST" FOR OUTPUT AS #1
DO
    INPUT "   NAME:       ", Name$  'Read entries from the keyboard.
    INPUT "   AGE:        ", Age$
    WRITE #1, Name$, Age$
    INPUT "Add another entry"; R$
LOOP WHILE UCASE$(R$) = "Y"
CLOSE #1
'Echo the file back.
OPEN "LIST" FOR INPUT AS #1
CLS
PRINT "Entries in file:": PRINT
DO WHILE NOT EOF(1)
    LINE INPUT #1, REC$  'Read entries from the file.
    PRINT REC$           'Print the entries on the screen.
LOOP
CLOSE #1
KILL "LIST"
```

See Also INKEY$ INPUT$ OPEN Statement File Modes

KEY(Assignment)
KEY { LIST | ON | OFF }

Assigns string values to function keys and, optionally, displays key values.

```
KEY key%, stringexpression$
KEY LIST
KEY ON
KEY OFF
```

_ key%	The number of a function key. Use 1 through 10 for function keys F1 through F10. Use 30 and 31 for function keys F11 and F12 on extended keyboards.
_ stringexpression$	A string of up to 15 characters that is returned when the function key is pressed.
_ LIST	Displays the assignments for each key.
_ ON	Turns on the function-key display line.
_ OFF	Turns off the function-key display line.

Example:

```
KEY 4, "MENU" + CHR$ (13)
KEY LIST
KEY 4, ""
KEY LIST
```

See Also KEY, ON KEY (Event Trapping)

LPOS()

Returns the number of characters sent to a printer since the last carriage return was sent.

```
LPOS(n%)

    _ n%    Indicates one of the printer ports:
              0 = LPT1, 1 = LPT1, 2 = LPT2, 3 = LPT3
Example:
    'This example requires a printer.
    LPRINT
    FOR i% = 1 TO 20
        LPRINT i%;
        IF LPOS(1) >= 10 THEN LPRINT     'Begin a new line.
    NEXT i%
```

OPEN (Communications)

Opens and initializes a communications channel for input or output (I/O). The OPEN COM statement must be executed before a device can be used for communication using an RS232 interface.

```
OPEN "COMn: optlist1 optlist2" [FOR mode] AS [#]filenum% [LEN=reclen%]
```

- n — The communications port to open (1 = COM1, 2 = COM2).
- optlist1 — The most-often-used communications parameters:
  [baud] [,[parity] [,[data] [,[stop]]]]
  baud is the baud rate of the device to be opened:
  75, 110, 150, 300, 600, 1200, 2400, 4800, 9600
  parity is the method of parity checking:
  N (none) E (even) O (odd)
  S (space) M (mark) PE (enable error checking)
  data is the number of data bits per byte:
  5, 6, 7, 8
  stop is the number of stop bits:
  1, 1.5, 2
  Defaults: 300 baud, even parity, 7 data bits, 1 stop bit.
- optlist2 — A list of less-often-used parameters, separated by commas:

Option	Description
ASC	Opens the device in ASCII mode.
BIN	Opens the device in binary mode.
CD[m]	Sets the timeout period (in milliseconds) on the Data Carrier Detect (DCD) line.
CS[m]	Sets the timeout period (in milliseconds) on the Clear to Send (CTS) line.
DS[m]	Sets the timeout period (in milliseconds) on the Data Set Ready (DS) line.
LF	Sends a line-feed character after a carriage return.
OP[m]	Specifies how long (in milliseconds) OPEN COM waits for all communications lines to become open.
RB[n]	Sets the size (in bytes) of the receive buffer.
RS	Suppresses detection of Request to Send (RTS).
TB[n]	Sets the size (in bytes) of the transmit buffer.

- mode — INPUT, OUTPUT, or RANDOM (the default).
  See OPEN Statement File Modes .
- filenum% — A number in the range 1 through 255 that identifies the communications channel as long as it is open.
- reclen% — Random-access-mode buffer size (default is 128 bytes).

Example:

```
'Use this example for trouble shooting serial communications problems.
'Slow baud, hardware handshaking is ignored and buffers are enlarged.
OPEN "COM1:300,N,8,1,CD0,CS0,DS0,OP0,RS,TB2048,RB2048" FOR RANDOM AS #1
```

Access Modes

The INPUT, OUTPUT, and RANDOM are used in the OPEN COM statement.

- INPUT specifies that the file is opened for sequential input.
- OUTPUT specifies that the file is opened for sequential output.
- RANDOM specifies that the file is opened in random-access file mode.
  RANDOM is the default file mode.

PRINT
LPRINT

PRINT writes data to the screen or to a file.
LPRINT prints data on the printer LPT1.

```
PRINT [#filenumber%,] [expressionlist] [{; | ,}]
LPRINT [expressionlist] [{; | ,}]
```

_ filenumber%	The number of an open file. If you don't specify a file number, PRINT writes to the screen.
_ expressionlist	A list of one or more numeric or string expressions to print.
_ {; | ,}	Determines where the next output begins: ; means print immediately after the last value. , means print at the start of the next print zone. Print zones are 14 characters wide.

Example:

```
OPEN "TEST.DAT" FOR OUTPUT AS #1
PRINT #1, USING "##.###  "; 12.12345
CLOSE
OPEN "TEST.DAT" FOR INPUT AS #1
INPUT #1, a$
PRINT a$
LPRINT "This is a line"; 1
LPRINT "This is a line",
LPRINT 2
```

See Also PRINT USING, LPRINT USING WIDTH WRITE

PRINT USING
LPRINT USING

PRINT USING writes formatted output to the screen or to a file.
LPRINT USING prints formatted output on the printer LPT1.

```
PRINT [#filenumber%,] USING formatstring$; expressionlist [{; | ,}]
LPRINT USING formatstring$; expressionlist [{; | ,}]
```

_ filenumber%	The number of an open sequential file.
_ formatstring$;	A string expression containing one or more format specifiers .
_ expressionlist	A list of one or more numeric or string expressions to print, separated by commas, semicolons, spaces, or tabs.
_ {; | ,}	Determines where the next output begins: ; means print immediately after the last value. , means print at the start of the next print zone. Print zones are 14 characters wide.

Example:

```
    a = 123.4567
    PRINT USING "###.##"; a
    LPRINT USING "+###.####"; a
    a$ = "ABCDEFG"
    PRINT USING "!"; a$
    LPRINT USING "\ \"; a$
```

See Also PRINT, LPRINT WIDTH

SCREEN()

Returns the ASCII value or color attribute of a character at a specified screen location.

SCREEN (row%,column% [,colorflag%])

_ row%	The row coordinate of a character.
_ column%	The column coordinate of a character.
_ colorflag%	A value (0 or 1) that specifies what is returned.

Value	Returns
0 (or omitted)	The character's ASCII code.
1	The character's color attribute.

Example:

```
CLS
PRINT "Hello"
PRINT "The ASCII value of character at 1,1 is"; SCREEN(1, 1)
```

See Also POINT SCREEN Statement
ASCII Character Codes Color Attributes and Values

SPC()

Skips a specified number of spaces in a PRINT or LPRINT statement.

SPC(n%)

_ n% The number of spaces to skip; a value in the range 0 through 32,767.

Example:

```
PRINT "Text1"; SPC(10); "Text2"
```

See Also PRINT, LPRINT, PRINT USING, LPRINT USING, SPACE$, TAB

TAB()

Moves the text cursor to a specified print position.

```
TAB(column%)

    _ column%    The column number of the new print position.

Example:
    PRINT TAB(25); "Text"

See Also     PRINT, LPRINT      PRINT USING, LPRINT USING
             SPC      SPACE$
```

VIEW PRINT

Sets the boundaries of the screen text viewport.

```
VIEW PRINT [toprow% TO bottomrow%]
```

- toprow% The number of the top row of the text viewport.
- bottomrow% The number of the bottom row of the text viewport.

- If you omit the toprow% and bottomrow% arguments, VIEW PRINT sets the entire screen as the text viewport.
- Ranges for toprow% and bottomrow% depend on the screen mode.

Example:

```
VIEW PRINT 10 TO 15
FOR i% = 1 TO 100      'Output will scroll.
    PRINT i%
NEXT i%
```

See Also CLS LOCATE PRINT, LPRINT SCREEN WIDTH
Screen Modes

WAIT

Suspends program execution until a specified bit pattern is input from an input port.

```
WAIT portnumber%, AND-expression% [,XOR-expression%]
```

- _ portnumber% — The number of the input port.
- _ AND-expression% — An integer expression that WAIT combines with the bit pattern value using an AND operator. When the result is nonzero, WAIT stops monitoring the port.
- _ XOR-expression% — Can be used to turn line bits on and off in the bit pattern before the AND operation is applied.

Example:

```
    'Reads the interrupt controller port address &H20.
    'Press any key to continue.
    WAIT &H20, 1
```

See Also INP, OUT Boolean Operators

WIDTH

Assign an output-line width to a device (such as a printer) or file, or change the number of screen-display columns and rows.

```
WIDTH [columns%] [,rows%]
WIDTH {#filenumber% | device$}, columns%
WIDTH LPRINT columns%
```

_ columns%	The desired width in columns. Screen display width must be 40 or 80 columns.
_ rows%	The desired screen-display height in rows. The value can be 25, 30, 43, 50, or 60, depending on your display adapter and screen mode.
_ #filenumber%	The number of an open file or device.
_ device$	The name of a device: SCRN:, COM1:, COM2:, LPT1:, LPT2:, LPT3:

Example:

```
OPEN "LPT1:" FOR OUTPUT AS #1
WIDTH #1, 132
```

See Also PRINT, LPRINT SCREEN VIEW PRINT

WRITE

Writes data to the screen or a sequential file.

```
WRITE [[#]filenumber%,] expressionlist
```

- _ filenumber% — The number of an open sequential file. If the file number is omitted, WRITE writes to the screen.
- _ expressionlist — One or more variables or expressions, separated by commas, whose values are written to the screen or file.

- _ WRITE inserts commas between items and quotation marks around strings as they are written. WRITE writes values to a file in a form that can be read by the INPUT statement.

Example:

```
CLS
OPEN "LIST" FOR OUTPUT AS #1
DO
    INPUT "   NAME:       ", Name$
    INPUT "   AGE:        ", Age$
    WRITE #1, Name$, Age$
    INPUT "Add another entry"; R$
LOOP WHILE UCASE$(R$) = "Y"
CLOSE #1
'Print the file to the screen.
OPEN "LIST" FOR INPUT AS #1
CLS
PRINT "Entries in file:": PRINT
DO WHILE NOT EOF(1)
    INPUT #1, Rec1$, Rec2$   'Read entries from file.
    PRINT Rec1$, Rec2$       'Print the entries on the screen.
LOOP
CLOSE #1
KILL "LIST"
```

See Also INPUT, LINE INPUT OPEN PRINT, LPRINT

3 Graphics Display

Content of section 3

CIRCLE

Draws a circle or ellipse on the screen.

```
CIRCLE [STEP] (x!,y!),radius![,[color%] [,[start!] [,[end!] [,aspect!]]]]
```

- _ STEP — Specifies that coordinates are relative to the current graphics cursor position.
- _ (x!,y!) — The coordinates for the center of the circle or ellipse.
- _ radius! — The radius of the circle or ellipse in the units of the current coordinate system, determined by the most recent SCREEN, VIEW, and WINDOW statements.
- _ color% — A color attribute that sets the circle's color. The available color attributes depend on your graphics adapter and the screen mode set by the most recent SCREEN statement.
- _ start! — The starting angle for the arc, in radians.
- _ end! — The ending angle for the arc, in radians.
- _ aspect! — The ratio of the length of the y axis to the length of the x axis, used to draw ellipses.

- _ To convert from degrees to radians, multiply degrees by (PI / 180).

Example:

```
'This example requires a color graphics adapter.
SCREEN 2
CIRCLE (320, 100), 200
CIRCLE STEP (0,0), 100
```

See Also COLOR DRAW LINE SCREEN VIEW WINDOW
Color Attributes and Values Screen Modes

COLOR

Sets the screen display colors.

```
COLOR [foreground%] [,[background%] [,border%]]    Screen mode 0 (text only)
COLOR [background%] [,palette%]                    Screen mode 1
COLOR [foreground%]                                Screen modes 4, 12, 13
COLOR [foreground%] [,background&]                 Screen modes 7-10
```

- `foreground%` / `foreground&` — A number that sets the foreground screen color. In screen mode 0, foreground% is a color attribute that sets the text color. In other screen modes, foreground% a color attribute or 4-bit color value (screen mode 4 only) that sets the text and line-drawing color.
- `background%` / `background&` — A number that sets the background screen color. In screen mode 0, background% is a color attribute. In screen mode 1, background% is a 4-bit color value. In screen modes 7-10, background& is a color value.
- `border%` — A color attribute that sets the screen border color.
- `palette%` — A number (0 or 1) that specifies which of two sets of color attributes to use:

palette%	Attribute 1	Attribute 2	Attribute 3
0	Green	Red	Brown
1	Cyan	Magenta	Bright white

- The available color attributes and values depend on your graphics adapter and the screen mode set by the most recent SCREEN statement.
- If your system is equipped with an EGA, VGA, or MCGA adapter, use the PALETTE statement to change the color assignments of color attributes.

Example:

```
'This example requires a color graphics adapter.
SCREEN 7
FOR i% = 0 TO 15
    COLOR i%
    PRINT i%
NEXT i%
```

See Also DRAW PAINT PALETTE, PALETTE USING SCREEN
Color Attributes and Values Screen Modes

DRAW

Draws an object.

```
DRAW commandstring$

    _ commandstring$    A string expression that contains one or more of
                        the following DRAW commands.

      Line-drawing and cursor-movement commands:
        D[n%]             Moves cursor down n% units.
        E[n%]             Moves cursor up and right n% units.
        F[n%]             Moves cursor down and right n% units.
        G[n%]             Moves cursor down and left n% units.
        H[n%]             Moves cursor up and left n% units.
        L[n%]             Moves cursor left n% units.
        M[{+|-}]x%,y%     Moves cursor to point x%,y%. If x% is preceded
                          by + or -, moves relative to the current point.
        R[n%]             Moves cursor right n% units.
        U[n%]             Moves cursor up n% units.
        [B]               Optional prefix that moves cursor without drawing.
        [N]               Optional prefix that draws and returns cursor to
                          its original position.

      Color, rotation, and scale commands:
        An%               Rotates an object n% * 90 degrees (n% can be 0, 1,
                          2, or 3).
        Cn%               Sets the drawing color (n% is a color attribute).
        Pn1%,n2%          Sets the paint fill and border colors of an object
                          (n1% is the fill-color attribute, n2% is the
                          border-color attribute).
        Sn%               Determines the drawing scale by setting the length
                          of a unit of cursor movement. The default n% is 4,
                          which is equivalent to 1 pixel.
        TAn%              Turns an angle n% degrees (-360 through 360).

    _ If you omit n% from line-drawing and cursor-movement commands, the
      cursor moves 1 unit.
    _ To execute a DRAW command substring from a DRAW command string, use
      the "X" command:

      DRAW "X"+ VARPTR$(commandstring$)

Example:
    'This example requires a color graphics adapter.
    SCREEN 1
    Triangle$ = "F60 L120 E60"
    DRAW "C2 X" + VARPTR$(Triangle$)
    DRAW "BD30 P1,2 C3 M-30,-30"

See Also    PALETTE, PALETTE USING    SCREEN    VARPTR$
            Color Attributes and Values
```

GET (Graphics Handling)
PUT (Graphics Handling)

GET captures a graphics screen image. PUT displays an image captured by GET.

```
GET [STEP](x1!,y1!)-[STEP](x2!,y2!), arrayname[(index%)]
PUT [STEP] (x1!,y1!), arrayname[(index%)] [,actionverb]
```

- STEP — Specifies that coordinates are relative to the current graphics cursor position.
- (x1!,y1!) — The upper-left coordinates of the image captured by GET or of the screen location where PUT displays the image.
- (x2!,y2!) — The lower-right coordinates of the captured image.
- arrayname — The name of the array where the image is stored. See Screen Image Arrays and Compatibility to determine the required size of the array.
- index% — The array index at which storage of the image begins.
- actionverb — A keyword indicating how the image is displayed:

Keyword	Action
AND	Merges stored image with an existing image.
OR	Superimposes stored image on existing image.
PSET	Draws stored image, erasing existing image.
PRESET	Draws stored image in reverse colors, erasing existing image.
XOR	Draws a stored image or erases a previously drawn image while preserving the background, producing animation effects.

- A PUT statement should always be executed in the same screen mode as the GET statement used to capture the image, or a compatible mode. See Screen Image Arrays and Compatibility .

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
DIM Box%(1 TO 200)
x1% = 0: x2% = 10: y1% = 0: y2% = 10
LINE (x1%, y1%)-(x2%, y2%), 2, BF
GET (x1%, y1%)-(x2%, y2%), Box%
DO
    PUT (x1%, y1%), Box%, XOR
    x1% = RND * 300
    y1% = RND * 180
    PUT (x1%, y1%), Box%
LOOP WHILE INKEY$ = ""
```

See Also SCREEN Screen Modes

LINE

Draws a line or rectangle on the screen.

```
LINE [[STEP](x1!,y1!)]-[STEP](x2!,y2!) [,[color%] [,[B | BF] [,style%]]]
```

_ STEP	Specifies that coordinates are relative to the current graphics cursor position.
_ (x1!,y1!), (x2!,y2!)	The screen coordinates of the start of the line and of the end of the line.
_ color%	A color attribute that sets the color of the line or rectangle. The available color attributes depend on your graphics adapter and the screen mode set by the most recent SCREEN statement.
_ B	Draws a rectangle instead of a line.
_ BF	Draws a filled box.
_ style%	A 16-bit value whose bits set whether or not pixels are drawn. Use to draw dashed or dotted lines.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
LINE (110, 70)-(190, 120), , B
LINE (0, 0)-(320, 200), 3, , &HFF00
```

See Also CIRCLE, INPUT, LINE INPUT, PRESET, PSET, SCREEN, Color Attributes and Values, Screen Modes

PAINT

Fills a graphics area with a specified color or pattern.

```
PAINT [STEP] (x!,y!)[,[{color% | tile$}] [,[bordercolor%] [,background$]]]
```

- STEP — Specifies that coordinates are relative to the current graphics cursor position.
- (x!,y!) — The screen coordinates where painting begins.
- color% — A color attribute that sets the fill color.
- tile$ — A fill pattern that is 8 bits wide and up to 64 bytes long, defined as follows:

```
tile$ = CHR$(arg1) + CHR$(arg2) + ... + CHR$(argn%)
```

  The arguments to CHR$ are numbers between 0 and 255. Each CHR$(argn%) defines a 1-byte, 8-pixel slice of the pattern based on the binary form of the number.
- bordercolor% — A color attribute that specifies the color of the filled area's border. PAINT stops filling an area when it encounters a border of the specified color.
- background$ — A 1-byte, 8-pixel background tile slice. Specifying a background tile slice allows you to paint over an area that has already been painted.

- The available color attributes depend on your graphics adapter and the screen mode set by the most recent SCREEN statement.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
CIRCLE (106, 100), 75, 1
LINE (138, 35)-(288, 165), 1, B
PAINT (160, 100), 2, 1
```

See Also ASC, CHR$ CIRCLE DRAW LINE SCREEN
Color Attributes and Values Screen Modes

PALETTE

Change the color assignments of color attributes in the current screen mode. PALETTE and PALETTE USING work only on systems equipped with EGA, VGA, or MCGA adapters.

```
PALETTE [attribute%,color&]
PALETTE USING arrayname#[(index%)]
```

- attribute% — The color attribute to change.
- color& — A color value to assign to an attribute.
- arrayname# — An array of color values to assign to the current screen mode's set of attributes. The array must be large enough to assign colors to all the attributes.
- index% — The index of the first array element to assign to an attribute.

- The available color attributes and values depend on your graphics adapter and the screen mode set by the most recent SCREEN statement.

Example:

```
'This example requires a color graphics adapter.
PALETTE 0, 1
SCREEN 1
FOR i% = 0 TO 3: a%(i%) = i%: NEXT i%
LINE (138, 35)-(288, 165), 3, BF
LINE (20, 10)-(160, 100), 2, BF
DO
    FOR i% = 0 TO 3
        a%(i%) = (a%(i%) + 1) MOD 16
    NEXT i%
    PALETTE USING a%(0)
LOOP WHILE INKEY$ = ""
```

See Also COLOR, SCREEN, Color Attributes and Values, Screen Modes

PCOPY

Copies one video memory page to another.

```
PCOPY sourcepage%,destinationpage%
```

- sourcepage% The number of a video memory page to copy.
- destinationpage% The number of the video memory page to copy to.

- The value that identifies the video page is determined by the size of video memory and the current screen mode.

Example:

```
PCOPY 1, 3
```

See Also SCREEN Screen Modes

PMAP()

Returns the window coordinate equivalent to a viewport coordinate, as defined by the WINDOW statement, or vice versa.

```
PMAP (startcoordinate#, n%)
```

- _ startcoordinate# A window or viewport coordinate.
- _ n% A value indicating which coordinate is returned:

startcoordinate#	n%	Returns
Window x coordinate	0	Viewport x coordinate
Window y coordinate	1	Viewport y coordinate
Viewport x coordinate	2	Window x coordinate
Viewport y coordinate	3	Window y coordinate

Example:

```
'This example requires a graphics adapter that supports screen mode 1.
SCREEN 1
WINDOW SCREEN (0, 0)-(100, 100)
PRINT "Logical x=50, physical x="; PMAP(50, 0)
PRINT "Logical y=50, physical y="; PMAP(50, 1)
```

See Also POINT VIEW WINDOW

POINT

Returns the current graphics cursor coordinates or the color attribute of a specified pixel.

```
POINT {(n%) | (x%,y%)}
```

_ (n%) Indicates the type of coordinate to return:

n%	Returns
0	The current viewport x coordinate
1	The current viewport y coordinate
2	The current window x coordinate
3	The current window y coordinate

_ (x%,y%) The coordinates of the pixel that POINT checks for color. If the coordinates are outside the current viewport, POINT returns -1.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
LINE (0, 0)-(100, 100), 2
LOCATE 14, 1
FOR y% = 1 TO 10
    FOR x% = 1 TO 10
        PRINT POINT(x%, y%);
    NEXT x%
    PRINT
NEXT y%
```

See Also COLOR PMAP SCREEN VIEW WINDOW
Color Attributes and Values

PRESET
PSET

Draw a specified point on the screen.

```
PRESET [STEP] (x!,y!) [,color%]
PSET [STEP] (x!,y!) [,color%]
```

- STEP — Specifies that the x! and y! are expressed relative to the current graphics cursor location.
- (x!,y!) — The screen coordinates of the pixel to be set.
- color% — A color attribute that sets the pixel color. If color% is omitted, PRESET uses the current background and PSET uses the current foreground color.

- Available color attributes depend on your graphics adapter and screen mode. Coordinate values depend on the graphics adapter, screen mode, and most recent VIEW and WINDOW statements.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
FOR i% = 0 TO 320
    PSET (i%, 100)
    FOR delay% = 1 TO 100: NEXT delay%
    PRESET (i%, 100)
NEXT i%
```

See Also SCREEN VIEW WINDOW
Color Attributes and Values Screen Modes

SCREEN

Sets the screen mode and other characteristics of your screen.

SCREEN mode% [,[colorswitch%] [,[activepage%] [,visualpage%]]]

- _ mode% — Sets the screen mode. See Screen Modes .
- _ colorswitch% — A value (0 or 1) that switches between color and monocolor display (modes 0 and 1 only):

Mode	Value	Action
0	0	Disables color
0	Nonzero	Enables color
1	0	Enables color
1	Nonzero	Disables color

- _ activepage% — The screen page that text or graphics output writes to.
- _ visualpage% — The screen page that is currently displayed on your screen.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1        '320 x 200 graphics
LINE (110, 70)-(190, 120), , B
LINE (0, 0)-(320, 200), 3, , &HFF00
```

See Also CIRCLE COLOR DRAW LINE PAINT SCREEN Function VIEW WINDOW Screen Modes

VIEW

Defines the size and location of a viewport where graphics can be displayed on the screen.

```
VIEW [[SCREEN] (x1!,y1!)-(x2!,y2!) [,[color%] [,border%]]]
```

_ SCREEN	Specifies that coordinates are relative to the screen rather than the viewport.
_ (x1!,y1!)-(x2!,y2!)	The coordinates of diagonally opposite corners of the viewport.
_ color%	A color attribute that sets the viewport fill color.
_ border%	A color attribute that sets the viewport border color.

- If all arguments are omitted, the entire screen is the viewport.
- The available color attributes depend on your graphics adapter and the screen mode set by the most recent SCREEN statement.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
VIEW (10, 10)-(300, 180), , 1
LOCATE 1, 11: PRINT "A big graphics viewport";
VIEW SCREEN (80, 80)-(200, 125), , 1
LOCATE 11, 11: PRINT "A small graphics viewport";
```

See Also CLS SCREEN VIEW PRINT WINDOW
Color Attributes and Values Screen Modes

WINDOW

Defines logical dimensions for the current graphics viewport. Use the WINDOW statement to define your own viewport coordinate system.

```
WINDOW [[SCREEN] (x1!,y1!)-(x2!,y2!)]
```

- _ SCREEN — Inverts the normal Cartesian direction of the y screen coordinates so that y values increase from the top of the screen to the bottom.
- _ (x1!,y1!) — Logical coordinates that map to the upper-left screen coordinates of the viewport.
- _ (x2!,y2!) — Logical coordinates that map to the lower-right screen coordinates of the viewport.

- _ WINDOW with no arguments disables the logical coordinate system.
- _ Use the VIEW statement to change the size of the viewport.

Example:

```
'This example requires a color graphics adapter.
SCREEN 1
FOR i% = 1 TO 10 STEP 2
    WINDOW (-160 / i%, -100 / i%)-(160 / i%, 100 / i%)
    CIRCLE (0, 0), 10
NEXT i%
```

See Also CLS PMAP POINT SCREEN VIEW WIDTH

4 Program Control

Content of section 4

CHAIN

Transfers control from the current program to another Basic program.

```
CHAIN filespec$
```

    _ filespec$ The name of the program to which control is passed.

```
Example:
    'Assumes the program TEST.BAS is in a \DOS directory.
    CHAIN "C:\DOS\TEST.BAS"
```

See Also CALL COMMON RUN Differences from BASICA

DO WHILE ... LOOP
DO UNTIL ... LOOP
DO ... LOOP WHILE
DO ... LOOP UNTIL

Repeats a block of statements while a condition is true or until a condition becomes true.

```
DO [{WHILE | UNTIL} condition]
    [statementblock]
LOOP

DO
    [statementblock]
LOOP [{WHILE | UNTIL} condition]
```

    _ condition   A numeric expression that Basic evaluates as true
                   (nonzero) or false (zero).

Example:

```
    i% = 0
    PRINT "Value of i% at beginning of loop is  "; i%
    DO WHILE i% < 10
        i% = i% + 1
    LOOP
    PRINT "Value of i% at end of loop is  "; i%
```

See Also EXIT FOR...NEXT WHILE...WEND

END
END {DEF | FUNCTION | IF | SELECT | SUB | TYPE}

Ends a program, procedure, block, or user-defined data type.

END [{DEF | FUNCTION | IF | SELECT | SUB | TYPE}]

- DEF — Ends a multiline DEF FN function definition.
- FUNCTION — Ends a FUNCTION procedure definition.
- IF — Ends a block IF...THEN...ELSE statement.
- SELECT — Ends a SELECT CASE block.
- SUB — Ends a SUB procedure.
- TYPE — Ends a user-defined data type definition.

- If no argument is supplied, END ends the program and closes all files.

Example:

```
PRINT "Game over."
END
```

See Also DEF FN, FUNCTION, IF...THEN...ELSE, SELECT CASE, STOP, SUB, SYSTEM, TYPE

EXIT

Exits a DO or FOR loop, a FUNCTION or SUB procedure, or a DEF FN function.

EXIT {DEF | DO | FOR | FUNCTION | SUB}

- DEF — Exits a DEF FN function.
- DO — Exits a DO loop.
- FOR — Exits a FOR loop.
- FUNCTION — Exits a FUNCTION procedure.
- SUB — Exits a SUB procedure.

Example:

```
i% = 0
DO
    i% = i% + 1
    IF i% = 500 THEN EXIT DO
LOOP
PRINT "EXIT at"; i%
```

See Also DEF FN DO...LOOP FOR...NEXT FUNCTION SUB

FOR ... NEXT

Repeats a block of statements a specified number of times.

```
FOR counter = start TO end [STEP increment]
    [statementblock]
NEXT [counter [,counter]...]
```

- counter — A numeric variable used as the loop counter.
- start and end — The initial and final values of the counter.
- increment — The amount the counter is changed each time through the loop.

Example:

```
FOR i% = 1 TO 15
    PRINT i%
NEXT i%
FOR i% = 7 to -6 STEP -3
    PRINT i%
NEXT i%
```

See Also DO...LOOP EXIT WHILE...WEND

IF THEN
ELSE
ELSEIF
ENDIF

Executes a statement or statement block depending on specified conditions.

```
IF condition1 THEN
    [statementblock-1]
[ELSEIF condition2 THEN
    [statementblock-2]]...
[ELSE
    [statementblock-n]]
END IF

IF condition THEN statements [ELSE statements]
```

_	condition1 condition2	Any expression that can be evaluated as true (nonzero) or false (zero).
_	statementblock-1 statementblock-2 statementblock-n	One or more statements on one or more lines.
_	statements	One or more statements, separated by colons.

Example:

```
    INPUT "1 or 2? ", i%
    IF i% = 1 OR i% = 2 THEN
        PRINT "OK"
    ELSE
        PRINT "Out of range"
    END IF
```

See Also ON...GOSUB ON...GOTO SELECT CASE

GOSUB ... RETURN

Branches to and returns from a subroutine.

```
GOSUB line1
.
.
.
RETURN [line2]
```

- line1 The label or line number of the first line of the subroutine.
- line2 The label or line number where the subroutine returns.

- If you don't supply a label or line number for RETURN, the program continues execution at the statement following the GOSUB (for subroutine calls) or where an event occurred (for event handling). See the ON keyword for information about event-handling statements.
- SUB and CALL statements provide a better alternative to GOSUB subroutines.

Example:

```
FOR i% = 1 TO 20
    GOSUB Square
NEXT i%
END

Square:
PRINT i%, i% * i%
RETURN
```

See Also CALL ON Keyword ON...GOSUB SUB

GOTO

Branches to a specified line.

GOTO line

- _ line The label or number of the line to execute next.

- _ DO...LOOP, SELECT CASE, IF...THEN...ELSE, SUB, and FUNCTION provide better ways to control the flow of your program.
- _ GOTO is also used as a keyword in the ON ERROR statement.

Example:

```
'Illustrates ERDEV, ERDEV$, ERL, ERR, ERROR, ON ERROR, and RESUME.
   ON ERROR GOTO Handler
10 CHDIR "a:\"                'Causes ERR 71 "Disk not ready"
                              'if no disk in Drive A.
20 y% = 0
30 x% = 5 / y%                'ERR 11 "Division by zero."
40 PRINT "x% ="; x%
50 ERROR 57                   'ERR 57 "Device I/O error."

Handler:
  PRINT
  PRINT "Error "; ERR; " on line "; ERL
  SELECT CASE ERR
    CASE 71
      PRINT "Using device "; ERDEV$; " device error code = "; ERDEV
      RESUME NEXT
    CASE 11
      INPUT "What value do you want to divide by"; y%
      RESUME                  'Retry line 30 with new value of y%.
    CASE ELSE
      PRINT "Unexpected error, ending program."
      END
  END SELECT
```

See Also DO...LOOP FUNCTION IF...THEN...ELSE
ON ERROR ON...GOTO SELECT CASE SUB

ON ... GOSUB ...
ON ... GOTO ...

Branch to one of several locations, depending on the value of an expression.

```
ON expression% GOSUB line-list
ON expression% GOTO line-list
```

_ expression%	An expression in the range 0 through 255.
_ line-list	A set of labels or line numbers. If the value of the expression is 1, the program branches to the first line in the list; if the expression is 2, it branches to the second line, and so on.

_ SELECT CASE provides a better way to perform multiple branching.

Example:

```
FOR i% = 1 TO 2
    ON i% GOSUB One, Two
NEXT i%
END

One: PRINT "One"
RETURN
Two: PRINT "Two"
RETURN
```

See Also SELECT CASE

SELECT CASE ... CASE ... END SELECT

Executes one of several statement blocks depending on the value of an expression.

```
SELECT CASE testexpression
CASE expressionlist1
  [statementblock-1]
[CASE expressionlist2
  [statementblock-2]]...
[CASE ELSE
  [statementblock-n]]
END SELECT
```

_ testexpression	Any numeric or string expression.
_ expressionlist1	One or more expressions to match testexpression.
expressionlist2	The IS keyword must precede any relational operators in an expression.
_ statementblock-1	One or more statements on one or more lines.
statementblock-2	
statementblock-n	

_ The expressionlist arguments can have any of these forms or a combination of them, separated by commas:

```
expression[,expression]...
expression TO expression
IS relational-operator expression
```

expression	Any numeric or string expression compatible with testexpression.
relational-operator	One of the following relational operators: <, <=, >, >=, <>, or =.

Example:

```
INPUT "Enter acceptable level of risk (1-5): ", Total
SELECT CASE Total

    CASE IS >= 5
        PRINT "Maximum risk and potential return."
        PRINT "Choose stock investment plan."

    CASE 2 TO 4
        PRINT "Moderate to high risk and potential return."
        PRINT "Choose mutual fund or corporate bonds."

    CASE 1
        PRINT "No risk, low return."
        PRINT "Choose IRA."

END SELECT
```

See Also IF...THEN...ELSE

STOP

Halts a program.

```
STOP
```

_ The STOP keyword also suspends trapping of events in these statements:

COM, ON COM	KEY, ON KEY	PEN, ON PEN
PLAY, ON PLAY	STRIG, ON STRIG	TIMER, ON TIMER

Example:

```
FOR i% = 1 TO 10
    PRINT i%
    IF i% = 5 THEN STOP     'STOP pauses; F5 Continues.
NEXT i%
```

See Also END SYSTEM

SYSTEM

Closes all open files and returns control to the operating system.

SYSTEM

See Also END STOP

WHILE ... WEND

Executes a series of statements as long as a specified condition is true.

```
WHILE condition
.
.
.
WEND
```

- condition — A numeric expression that Basic evaluates as true (nonzero) or false (zero).

- DO...LOOP provides a better way to execute statements in a program loop.

See Also DO...LOOP FOR...NEXT

5 Procedures and Functions

Content of section 5

CALL

Transfers control to a SUB procedure.

[CALL] name [([argumentlist])]

- name — The name of the SUB procedure to call.
- argumentlist — The variables or constants to pass to the SUB procedure. Separate multiple arguments with commas. Specify array arguments with the array name followed by empty parentheses.

- If you omit the CALL keyword, also omit the parentheses around argumentlist. Either declare the procedure in a DECLARE statement before calling it, or save the program and QBasic automatically generates a DECLARE statement.
- To specify an argument whose value will not be changed by the procedure, enclose the argument in parentheses.

Example:

The program REMLINE.BAS illustrates calling SUB procedures. To view or run this program, load REMLINE.BAS using the Open command from the File menu.

See Also CALL ABSOLUTE DECLARE SUB

COMMON

Defines global variables that can be shared throughout a program or between chained programs.

```
COMMON [SHARED] variablelist
```

- _ SHARED — Indicates that variables are shared with all SUB or FUNCTION procedures.
- _ variablelist — One or more variables to be shared:

```
variable[( )] [AS type] [, variable[( )] [AS type]]...
```

  - variable — A Basic variable name. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
  - type — The data type of the variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type).

- _ Unless it has been declared as a static array in a preceding DIM statement, an array variable in a COMMON statement is a dynamic array. Its dimensions must be set in a later DIM or REDIM statement.

See Also CHAIN, DIM, REDIM, FUNCTION, SHARED, STATIC, SUB

DECLARE {FUNCTION | SUB}

Declares a FUNCTION or SUB procedure and invokes argument data type checking.

DECLARE {FUNCTION | SUB} name [([parameterlist])]

- name — The name of the procedure.
- parameterlist — One or more variables that specify parameters to be passed to the procedure when it is called:

  variable[( )] [AS type] [, variable[( )] [AS type]]...

  variable — A Basic variable name.
  type — The data type of the variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type). ANY allows any data type.

- DECLARE is required if you call SUB procedures without CALL. QBasic automatically generates DECLARE statements when you save your program.

Example:
The program REMLINE.BAS illustrates declaring FUNCTION and SUB procedures. To view or run this program, load REMLINE.BAS using the Open command from the File menu.

See Also CALL FUNCTION SUB

DEF FN

Defines a function.

```
DEF FNname[(parameterlist)] = expression
DEF FNname[(parameterlist)]
    [statementblock]
  FNname = expression
    [statementblock]
  EXIT DEF]
    [statementblock]
  END DEF
```

_ parameterlist One or more arguments in the following form:

```
variable[( )] [AS type] [, variable[( )] [AS type]]...
```

```
variable    A Basic variable name.
type        The data type of the variable (INTEGER,
            LONG, SINGLE, DOUBLE, STRING, or a
            user-defined data type).
```

_ expression The return value of the function.

_ The FUNCTION statement provides a better way to define a function.

See Also EXIT FUNCTION SHARED, STATIC

END {DEF | FUNCTION | IF | SELECT | SUB | TYPE}

Ends a procedure, block, or user-defined data type.

END [{DEF | FUNCTION | IF | SELECT | SUB | TYPE}]

- DEF — Ends a multiline DEF FN function definition.
- FUNCTION — Ends a FUNCTION procedure definition.
- IF — Ends a block IF...THEN...ELSE statement.
- SELECT — Ends a SELECT CASE block.
- SUB — Ends a SUB procedure.
- TYPE — Ends a user-defined data type definition.

- If no argument is supplied, END ends the program and closes all files.

Example:

```
PRINT "Game over."
END
```

See Also DEF FN, FUNCTION, IF...THEN...ELSE, SELECT CASE, STOP, SUB, SYSTEM, TYPE

EXIT

Exits a DO or FOR loop, a FUNCTION or SUB procedure, or a DEF FN function.

EXIT {DEF | DO | FOR | FUNCTION | SUB}

- DEF — Exits a DEF FN function.
- DO — Exits a DO loop.
- FOR — Exits a FOR loop.
- FUNCTION — Exits a FUNCTION procedure.
- SUB — Exits a SUB procedure.

Example:

```
i% = 0
DO
    i% = i% + 1
    IF i% = 500 THEN EXIT DO
LOOP
PRINT "EXIT at"; i%
```

See Also DEF FN DO...LOOP FOR...NEXT FUNCTION SUB

FUNCTION

Defines a FUNCTION procedure.

```
FUNCTION name [(parameterlist)] [STATIC]
    [statementblock]
  name = expression
    [statementblock]
END FUNCTION
```

- _ name — The name of the function and the data type it returns, specified by a data-type suffix (%, &, !, #, or $).
- _ parameterlist — One or more variables that specify parameters to be passed to the function when it is called:

  ```
  variable[( )] [AS type] [, variable[( )] [AS type]]...
  ```

  variable — A Basic variable name.
  type — The data type of the variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type).

- _ STATIC — Specifies that the values of the function's local variables are saved between function calls.
- _ expression — The return value of the function.

- _ When you call the function, you can specify that an argument's value will not be changed by the function by enclosing the argument in parentheses.

Example:
The program REMLINE.BAS illustrates calling FUNCTION procedures. To view or run this program, load REMLINE.BAS using the Open command from the File menu.

See Also DECLARE DEF FN EXIT SHARED, STATIC SUB

RUN

Runs the current program or a specified program.

```
RUN [{linenumber | file$}]
```

- _ linenumber — The line number in the current program where execution should begin. If no line number is specified, execution begins at the first executable line.
- _ file$ — The name of a Basic source file. QBasic assumes a .BAS extension.

- _ RUN closes all files and clears program memory before loading a program. Use the CHAIN statement to run a program without closing open files.

Example:

```
'Assumes the program TEST.BAS is in a \DOS directory.
RUN "C:\DOS\TEST.BAS"
```

See Also CHAIN

SHARED
STATIC

SHARED gives procedures access to module-level variables.
STATIC makes a variable local to a function or procedure and preserves its value between calls.

```
SHARED variable[()] [AS type] [,variable[()] [AS type]]...
STATIC variable[()] [AS type] [,variable[()] [AS type]]...
```

_ variable	The name of the module-level variable to share or variable to make static. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
_ AS type	Declares the data type of the variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined type).

Example:
The program REMLINE.BAS illustrates using the SHARED and STATIC statements. To view or run this program, load REMLINE.BAS using the Open command from the File menu.

See Also COMMON DIM, REDIM

SUB

Defines a SUB procedure.

```
SUB name[(parameterlist)] [STATIC]
  [statementblock]
END SUB
```

- _ name — The name of the SUB procedure, up to 40 characters long, with no data type suffix.
- _ parameterlist — One or more variables that specify parameters to be passed to the SUB procedure when it is called:

  ```
  variable[( )] [AS type] [, variable[( )] [AS type]]...
  ```

  variable — A Basic variable name.
  type — The data type of the variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, or a user-defined data type).

- _ STATIC — Specifies that the values of the SUB procedure's local variables are saved between function calls.

- _ When you call the SUB procedure, you can specify that an argument's value will not be changed by the procedure by enclosing the argument in parentheses.

Example:
The program REMLINE.BAS illustrates calling SUB procedures. To view or run this program, load REMLINE.BAS using the Open command from the File menu.

See Also CALL DECLARE EXIT FUNCTION SHARED, STATIC

6 File Access Handling

Content of section 6

CLOSE

Closes one or more open files or devices.

```
CLOSE [[#]filenumber%[,[#]filenumber%]...]
```

- filenumber% The number of an open file or device.
- CLOSE with no arguments closes all open files and devices.

Example:

```
CLS
INPUT "Enter filename: ", n$
OPEN n$ FOR OUTPUT AS #1
PRINT #1, "This is saved to the file."
CLOSE
OPEN n$ FOR INPUT AS #1
INPUT #1, a$
PRINT "Read from file: "; a$
CLOSE
```

See Also END OPEN RESET STOP

EOF()

Tests for the end of a file. EOF returns true (nonzero) if the end of a file has been reached.

```
EOF(filenumber%)

    _ filenumber%    The number of an open file.
```

Example:

```
CLS
OPEN "TEST.DAT" FOR OUTPUT AS #1
FOR i% = 1 TO 10
    WRITE #1, i%, 2 * i%, 5 * i%
NEXT i%
CLOSE #1
OPEN "TEST.DAT" FOR INPUT AS #1
DO
    LINE INPUT #1, a$
    PRINT a$
LOOP UNTIL (EOF(1))
```

See Also CLOSE LOC LOF OPEN

FIELD

Allocates space for variables in a random-access file buffer.

```
FIELD [#]filenumber%, fieldwidth% AS stringvariable$
                    [,fieldwidth% AS stringvariable$] ...
```

- filenumber% The number of an open file.
- fieldwidth% The number of characters in the field.
- stringvariable$ A variable that identifies the field and contains field data.

- Record variables usually provide a better way to handle record data.

Example:

```
OPEN "FILEDAT.DAT" FOR RANDOM AS #1 LEN = 80
FIELD #1, 30 AS name$, 50 AS address$
```

See Also GET, PUT LSET, RSET TYPE Differences from BASICA

FILEATTR()

Returns information about an open file.

FILEATTR(filenumber%,attribute%)

- _ filenumber% The number of an open file.
- _ attribute% Specifies the type of information to return. When attribute% is 1, FILEATTR returns a value indicating the file's access mode:

Value	Mode
1	Input
2	Output
4	Random
8	Append
32	Binary

When attribute% is 2, FILEATTR returns the DOS file handle.

Example:

```
OPEN "TEST.DAT" FOR BINARY AS #1
PRINT FILEATTR(1, 1)
CLOSE
```

See Also OPEN

FREEFILE

Returns the next valid unused file number.

FREEFILE

Example:

```
    OPEN "TEST.DAT" FOR OUTPUT AS #1
    PRINT "Next file number: "; FREEFILE
    CLOSE
```

See Also OPEN

GET (File Handling)
PUT (File Handling)

GET reads from a file into a random-access buffer or variable.
PUT writes a variable or random-access buffer to a file.

```
GET [#]filenumber%[,[recordnumber&][,variable]]
PUT [#]filenumber%[,[recordnumber&][,variable]]
```

- filenumber% — The number of an open file.
- recordnumber& — For random-access files, the number of the record to read or write. For binary-mode files, the byte position where reading or writing starts.
- variable — For GET, a variable used to receive input from the file. For PUT, a variable that contains output to write to the file. The variable is usually a variable of a user-defined data type.

Example:

```
TYPE TestRecord
    Student AS STRING * 20
    Score AS SINGLE
END TYPE
DIM MyClass AS TestRecord
OPEN "FINAL.DAT" FOR RANDOM AS #1 LEN = LEN(MyClass)
MyClass.Student = "MarySa"
MyClass.Score = 99
PUT #1, 1, MyClass
CLOSE #1
OPEN "FINAL.DAT" FOR RANDOM AS #1 LEN = LEN(MyClass)
GET #1, 1, MyClass
PRINT "STUDENT:", MyClass.Student
PRINT "SCORE:", MyClass.Score
CLOSE #1
KILL "FINAL.DAT"
```

See Also FIELD LSET, RSET MKn$, CVn Functions TYPE

INPUT
LINE INPUT

INPUT reads input from the keyboard or a file. LINE INPUT reads a line of up to 255 characters from the keyboard or a file.

```
INPUT [;] ["prompt"{; | ,}] variablelist
LINE INPUT [;] ["prompt";] variable$
INPUT #filenumber%, variablelist
LINE INPUT #filenumber%, variable$
```

- prompt — An optional literal string that is displayed before the user enters data. A semicolon after prompt appends a question mark to the prompt string.
- variablelist — One or more variables, separated by commas, in which data entered from the keyboard or read from a file is stored. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and period (.).
- variable$ — Holds a line of characters entered from the keyboard or read from a file.
- filenumber% — The number of an open file.

- INPUT uses a comma as a separator between entries. LINE INPUT reads all characters up to a carriage return.
- For keyboard input, a semicolon immediately after INPUT keeps the cursor on the same line after the user presses the Enter key.

Example:

```
CLS
OPEN "LIST" FOR OUTPUT AS #1
DO
    INPUT "   NAME:       ", Name$  'Read entries from the keyboard.
    INPUT "   AGE:        ", Age$
    WRITE #1, Name$, Age$
    INPUT "Add another entry"; R$
LOOP WHILE UCASE$(R$) = "Y"
CLOSE #1
'Echo the file back.
OPEN "LIST" FOR INPUT AS #1
CLS
PRINT "Entries in file:": PRINT
DO WHILE NOT EOF(1)
    LINE INPUT #1, REC$  'Read entries from the file.
    PRINT REC$           'Print the entries on the screen.
LOOP
CLOSE #1
KILL "LIST"
```

See Also INKEY$ INPUT$ OPEN Statement File Modes

INPUT$

Returns a string of characters read from a specified file.

```
INPUT$(n[,[#]filenumber%])

    _ n              The number of characters (bytes) to read.
    _ filenumber%    The number of an open file. If filenumber% is
                     omitted, INPUT$ reads from the keyboard.

Example:
    OPEN "TEST.DAT" FOR OUTPUT AS #1
    PRINT #1, "The text"
    CLOSE
    OPEN "TEST.DAT" FOR INPUT AS #1
    PRINT INPUT$(3, 1)        'Print first 3 characters.
    CLOSE

See Also    INPUT, LINE INPUT
```

LOC()

Returns the current position within a file.

LOC(filenumber%)

- filenumber% The number of an open file or device.

- For binary files, LOC returns the position of the last byte read or written.
- For random-access files, LOC returns the number of the last record read from or written to the file.
- For sequential files, LOC returns the current byte position in the file, divided by 128.

Example:

```
OPEN "TEST.DAT" FOR RANDOM AS #1
FOR i% = 1 TO 10
   PUT #1, , i%
NEXT i%
SEEK #1, 2
GET #1, , i%
PRINT "Data: "; i%; " Current record: "; LOC(1); " Next: "; SEEK(1)
```

See Also EOF SEEK

LOCK
UNLOCK

LOCK limits or prevents access to a file by a network process.
UNLOCK releases the locks imposed by the last LOCK statement.

```
LOCK [#]filenumber% [,{record& | [start&] TO end&}]
UNLOCK [#]filenumber% [,{record& | [start&] TO end&}]
```

_ filenumber%	The number of an open file.
_ record&	For random-access files, the number of a record to lock, relative to the first record in the file. For binary files, the number of a byte to lock, relative to the first byte in the file.
_ start& and end&	The numbers of the first and last records or bytes in a range of records or bytes to lock or unlock.

_ For sequential files, LOCK and UNLOCK affect the entire file.

Example:

```
'This example runs only in a network environment.
OPEN "TEST.DAT" FOR RANDOM AS #1
FOR i% = 1 TO 10
    PUT #1, , i%
NEXT i%
LOCK #1, 2          'Lock record 2.
GET #1, 2, i%
UNLOCK #1, 2        'Unlock record 2.
```

Returns the length of a file in bytes.

```
LOF(filenumber%)
```

_ filenumber%	The number of an open file.

Example:

```
INPUT "Enter filename: "; f$
OPEN f$ FOR BINARY AS #1
PRINT "File length = "; LOF(1)
CLOSE
```

LOF()

Returns the length of a file in bytes.

```
LOF(filenumber%)

    _ filenumber%    The number of an open file.

Example:
    INPUT "Enter filename: "; f$
    OPEN f$ FOR BINARY AS #1
    PRINT "File length = "; LOF(1)
    CLOSE
```

LSET
RSET

LSET and RSET move data into a random-access file buffer (in preparation for a PUT statement) and left- or right-justify the value of a string variable. LSET also copies the contents of one record variable to another.

```
LSET stringvariable$=stringexpression$
RSET stringvariable$=stringexpression$
LSET recordvariable1=recordvariable2
```

_ stringvariable$	Any string variable or a random-access file field defined in a FIELD statement.
_ stringexpression$	For LSET, the left-justified version of stringvariable$. For RSET, the right-justified version of stringvariable$.
_ recordvariable1 recordvariable2	Record variables of any user-defined data type. Use LSET to assign a record variable of one data type to a different user-defined data type.

Example:

```
OPEN "FILEDAT.DAT" FOR RANDOM AS #1 LEN = 10
FIELD #1, 5 AS Ls1$, 5 AS Rs1$
LSET Ls1$ = "LSET"
RSET Rs1$ = "RSET"
PUT #1, 1
CLOSE #1
OPEN "FILEDAT.DAT" FOR RANDOM AS #1 LEN = 10
FIELD #1, 5 AS Ls2$, 5 AS Rs2$
GET #1, 1
PRINT "*" + Ls2$ + "*", "*" + Rs2$ + "*"
CLOSE #1
```

See Also FIELD GET, PUT

MKI$() CVI()
MKL$() CVL()
MKS$() CVS()
MKD$() CVD()

MKI$, MKL$, MKS$, and MKD$ convert numbers to numeric strings that can be stored in FIELD statement string variables. CVI, CVL, CVS, and CVD convert those strings back to numbers.

```
MKI$(integer-expression%)
MKL$(long-integer-expression&)
MKS$(single-precision-expression!)
MKD$(double-precision-expression#)
CVI(2-byte-numeric-string)
CVL(4-byte-numeric-string)
CVS(4-byte-numeric-string)
CVD(8-byte-numeric-string)
```

Function	Returns	Function	Returns
MKI$	A 2-byte string	CVI	An integer
MKL$	A 4-byte string	CVL	A long integer
MKS$	A 4-byte string	CVS	A single-precision number
MKD$	An 8-byte string	CVD	A double-precision number

See Also FIELD MKSMBF$, MKDMBF$, CVSMBF, CVDMBF

MKSMBF$() CVSMBF ()
MKDMBF$() CVDMBF ()

MKSMBF$ and MKDMBF$ convert IEEE-format numbers to Microsoft-Binary-format numeric strings that can be stored in FIELD statement string variables. CVSMBF and CVDMBF convert those strings back to IEEE-format numbers.

```
MKSMBF$(single-precision-expression!)
MKDMBF$(double-precision-expression#)
CVSMBF (4-byte-numeric-string)
CVDMBF (8-byte-numeric-string)
```

Function	Returns
MKSMBF$	A 4-byte string containing a Microsoft-Binary-format number
MKDMBF$	An 8-byte string containing a Microsoft-Binary-format number
CVSMBF	A single-precision number in IEEE format
CVDMBF	A double-precision number in IEEE format

_ These functions are useful for maintaining data files created with older versions of Basic.

Example:

```
TYPE Buffer
    SngNum AS STRING * 4
    DblNum AS STRING * 8
END TYPE
DIM RecBuffer AS Buffer
OPEN "TESTDAT.DAT" FOR RANDOM AS #1 LEN = 12
SNum = 98.9
DNum = 645.3235622#
RecBuffer.SngNum = MKSMBF$(SNum)
RecBuffer.DblNum = MKDMBF$(DNum)
PUT #1, 1, RecBuffer
GET #1, 1, RecBuffer
CLOSE #1
PRINT CVSMBF(RecBuffer.SngNum), CVDMBF(RecBuffer.DblNum)
```

See Also FIELD MKn$, CVn

OPEN (File Handling)

Opens a file or device.

```
OPEN file$ [FOR mode] [ACCESS access] [lock] AS [#]filenumber% [LEN=reclen%]
```

- _ file$ The name of the file or device. The file name may include a drive and path.
- _ mode One of the following file modes: APPEND, BINARY, INPUT, OUTPUT, or RANDOM. See OPEN Statement File Modes .
- _ access In network environments, specifies whether the file is opened for READ, WRITE, or READ WRITE access. See OPEN Statement ACCESS Clause .
- _ lock Specifies the file locking in network environments: SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE.
- _ filenumber% A number in the range 1 through 255 that identifies the file while it is open.
- _ reclen% For random-access files, the record length (default is 128 bytes). For sequential files, the number of characters buffered (default is 512 bytes).

Example:

```
INPUT "Enter Filename: "; n$
OPEN n$ FOR OUTPUT AS #1
PRINT #1, "This is saved to the file."
CLOSE
OPEN n$ FOR INPUT AS #1
INPUT #1, a$
PRINT "Read from file: "; a$
CLOSE
```

Alternate Syntax (legacy compatability)

```
OPEN mode2$,[#]filenum%,file$[,reclen%]
```

- _ mode2$ A string expression that begins with one of the following characters and specifies the file mode:
  - O Sequential output mode.
  - I Sequential input mode.
  - R Random-access file input/output mode.
  - B Binary-file mode.
  - A Sequential output mode. Sets the file pointer to the end of the file and the record number to the last record of the file. A PRINT # or WRITE # statement extends (appends to) the file.
- _ filenum% A number in the range 1 through 255 that identifies the file while it is open.
- _ file$ The name of the file (may include drive and path).
- _ reclen% For random-access files, the record length in bytes. For sequential files, the number of characters buffered.

- _ QBasic supports this syntax for compatibility with programs written in earlier versions of BASIC.

Access Modes

The APPEND, BINARY, INPUT, OUTPUT, and RANDOM keywords are used in the OPEN statement to specify file modes.

- APPEND specifies that the file is to be opened for sequential output and sets the file pointer to the end of the file. A PRINT # or WRITE # statement then extends (appends to) the file.
- BINARY specifies a binary-file mode. In binary mode, you can read or write information to any byte position in the file using GET or PUT statements.
- INPUT specifies that the file is opened for sequential input.
- OUTPUT specifies that the file is opened for sequential output.
- RANDOM specifies that the file is opened in random-access file mode. RANDOM is the default file mode.

ACCESS {READ | WRITE | READ WRITE}

Specifies the type of access users have to an opened file.

ACCESS {READ | WRITE | READ WRITE}

- READ — Opens a file for reading only.
- WRITE — Opens a file for writing only.
- READ WRITE — Opens a file for both reading and writing. READ WRITE mode is valid only for random-access and binary-mode files, and files opened for APPEND (sequential access).

See Also CLOSE FREEFILE OPEN COM

RESET

Closes all open files and devices.

RESET

See Also CLOSE END OPEN STOP

SEEK()
SEEK

```
The SEEK function returns the current file position.
The SEEK statement sets the file position for the next read or write.

SEEK(filenumber%)
SEEK [#]filenumber%, position&

    _ filenumber%    The number of an open file.
    _ position&      The position where the next read or write occurs. For
                     random-access files, a record number. For other files,
                     the byte position relative to the beginning of the file.
                     The first byte is at position 1.

Example:
    OPEN "TEST.DAT" FOR RANDOM AS #1
    FOR i% = 1 TO 10
        PUT #1, , i%
    NEXT i%
    SEEK #1, 2
    GET #1, , i%
    PRINT "Data: "; i%; " Current record: "; LOC(1); " Next: "; SEEK(1)

See Also     GET, PUT      OPEN
```

WRITE

Writes data to the screen or a sequential file.

```
WRITE [[#]filenumber%,] expressionlist
```

- _ filenumber% — The number of an open sequential file. If the file number is omitted, WRITE writes to the screen.
- _ expressionlist — One or more variables or expressions, separated by commas, whose values are written to the screen or file.

- _ WRITE inserts commas between items and quotation marks around strings as they are written. WRITE writes values to a file in a form that can be read by the INPUT statement.

Example:

```
CLS
OPEN "LIST" FOR OUTPUT AS #1
DO
    INPUT "   NAME:       ", Name$
    INPUT "   AGE:        ", Age$
    WRITE #1, Name$, Age$
    INPUT "Add another entry"; R$
LOOP WHILE UCASE$(R$) = "Y"
CLOSE #1
'Print the file to the screen.
OPEN "LIST" FOR INPUT AS #1
CLS
PRINT "Entries in file:": PRINT
DO WHILE NOT EOF(1)
    INPUT #1, Rec1$, Rec2$   'Read entries from file.
    PRINT Rec1$, Rec2$       'Print the entries on the screen.
LOOP
CLOSE #1
KILL "LIST"
```

See Also INPUT, LINE INPUT OPEN PRINT, LPRINT

7 Arithmetic Operations

Content of section 7

ABS()
SGN()

ABS returns the absolute value of a number.
SGN returns a value indicating the sign of a numeric expression (1 if the expression is positive, 0 if it is zero, or -1 if it is negative).

```
ABS(numeric-expression)
SGN(numeric-expression)
```

    _ numeric-expression   Any numeric expression.

Example:

```
    PRINT ABS(45.5 - 100!)           'Output is:  54.5
    PRINT SGN(1), SGN(-1), SGN(0)    'Output is:  1  -1  0
```

EXP()
LOG()

EXP returns e raised to a specified power, where e is the base of natural logarithms. LOG returns the natural logarithm of a numeric expression.

```
EXP(numeric-expression)
LOG(numeric-expression)
```

_ numeric-expression	For EXP, a number less than or equal to 88.02969. For LOG, any positive numeric expression.

Example:

```
    PRINT EXP(0), EXP(1)          'Output is:  1  2.718282
    PRINT LOG(1), LOG(EXP(1))     'Output is:  0  1
```

FIX()
INT()

FIX truncates a floating-point expression to its integer portion.
INT returns the largest integer less than or equal to a numeric expression.

```
FIX(numeric-expression)
INT(numeric-expression)
```

    _ numeric-expression   Any numeric expression.

Example:

```
    PRINT FIX(12.49), FIX(12.54)    'Output is:  12  12
    PRINT INT(12.54), INT(-99.4)    'Output is:  12  -100
```

See Also   CINT, CLNG

MOD

Divides one number by another and returns the remainder.

```
numeric-expression1 MOD numeric-expression2
```

_	numeric-expression1 numeric-expression2	Any numeric expressions. Real numbers are rounded to integers.

Example:

```
PRINT 19 MOD 6.7    'QBasic rounds 6.7 to 7, then divides.
                    'Output is:  5
```

RANDOMIZE
RND()

RANDOMIZE initializes the random-number generator.
RND returns a single-precision random number between 0 and 1.

```
RANDOMIZE [seed%]
RND[(n#)]
```

- _ seed% A number used to initialize the random-number generator. If omitted, RANDOMIZE prompts for it.
- _ n# A value that sets how RND generates the next random number:

n#	RND returns
Less than 0	The same number for any n#
Greater than 0 (or omitted)	The next random number
0	The last number generated

Example:

```
RANDOMIZE TIMER
x% = INT(RND * 6) + 1
y% = INT(RND * 6) + 1
PRINT "Roll of two dice: die 1 ="; x%; "and die 2 ="; y%
```

SQR()

Returns the square root of a numeric expression.

SQR(numeric-expression)

_ numeric-expression A value greater than or equal to zero.

Example:

```
PRINT SQR(25), SQR(2)    'Output is:  5  1.414214
```

In a FOR...NEXT loop, specifies how much to increase the counter in each iteration. In graphics statements, specifies that pixel coordinates are relative to the current graphics cursor position.

See Also CIRCLE FOR...NEXT GET, PUT LINE
PAINT PRESET, PSET

TIMER

Returns the number of seconds elapsed since midnight.

TIMER

- Use TIMER to time programs or parts of programs, or with the RANDOMIZE statement to seed the random-number generator.

Example:

```
RANDOMIZE TIMER
```

See Also RANDOMIZE, RND TIMER, ON TIMER Statements

Trigonometric Functions
ATN() COS() SIN() TAN()

ATN returns the arctangent of a specified numeric expression.
COS, SIN, and TAN return the cosine, sine, and tangent of a specified angle.

```
ATN(numeric-expression)
COS(angle)
SIN(angle)
TAN(angle)
```

- numeric-expression The ratio between the sides of a right triangle.
- angle An angle expressed in radians.

- The ATN function returns an angle in radians.
- To convert from degrees to radians, multiply degrees by (PI / 180).

Example:

```
CONST PI=3.141592654
PRINT ATN(TAN(PI/4.0)), PI/4.0    'Output is:  .7853981635  .7853981635
PRINT (COS(180 * (PI / 180)))     'Output is:  -1
PRINT (SIN(90 * (PI / 180)))      'Output is:  1
PRINT (TAN(45 * (PI / 180)))      'Output is:  1.000000000205103
```

8 String Operations

Content of section 8

DATE$

```
The DATE$ function returns the computer's current system date.
The DATE$ statement sets the current system date on your computer.

DATE$
DATE$ = stringexpression$

    _ stringexpression$    The date in one of the following forms:
                           mm-dd-yy, mm-dd-yyyy, mm/dd/yy, mm/dd/yyyy.

    _ The DATE$ function returns a string in the form mm-dd-yyyy.

Example:
    PRINT DATE$
    DATE$ = "01-01-90"    'Note: The new system date remains in effect until
                          '      you change it again.
    PRINT "Date set to "; DATE$

See Also    TIME$
```

HEX$()
OCT$()

```
HEX$ returns a hexadecimal string representation of a number.
OCT$ returns an octal string representation of a number.

HEX$(numeric-expression&)
OCT$(numeric-expression&)

    _ numeric-expression&    Any numeric expression. The expression is
                             rounded to an integer or long integer before it
                             is evaluated.

Example:
    INPUT x
    a$ = HEX$ (x)
    b$ = OCT$ (x)
    PRINT x; "decimal is "; a$; " hexadecimal and "; b$; " in octal."
```

INSTR()

Returns the position of the first occurrence of a string in another string.

```
INSTR([start%,]stringexpression1$,stringexpression2$)
```

_ start%	Sets the character position where the search begins. If start% is omitted, INSTR starts at position 1.
_ stringexpression1$	The string to search.
_ stringexpression2$	The string to look for.

Example:

```
a$ = "Microsoft QBasic"
PRINT "String position ="; INSTR(1, a$, "QBasic")
```

See Also LEFT$, RIGHT$ LEN MID$

Specify the data type for a variable in a declarative statement or parameter list:

_ INTEGER	A 16-bit signed integer variable.
_ LONG	A 32-bit signed integer variable.
_ SINGLE	A single-precision 32-bit floating-point variable.
_ DOUBLE	A double-precision 64-bit floating-point variable.
_ STRING * n%	A fixed-length string variable n% bytes long.
_ STRING	A variable-length string variable.

See Also

AS	Basic Character Set	COMMON
DECLARE	DEF FN	DIM, REDIM
FUNCTION	SHARED, STATIC	SUB
TYPE		

LCASE$()
UCASE$()

Convert strings to all lowercase or all uppercase letters.

```
LCASE$(stringexpression$)
UCASE$(stringexpression$)

    _ stringexpression$    Any string expression.
```

Example:

```
    Test$ = "THE string"
    PRINT Test$
    PRINT LCASE$(Test$); " in lowercase"
    PRINT UCASE$(Test$); " IN UPPERCASE"
```

Return a specified number of leftmost or rightmost characters in a string.

```
LEFT$(stringexpression$,n%)
RIGHT$(stringexpression$,n%)

    _ stringexpression$    Any string expression.
    _ n%                   The number of characters to return, beginning
                           with the leftmost or rightmost string character.
```

Example:

```
    a$ = "Microsoft QBasic"
    PRINT LEFT$(a$, 5)     'Output is:  Micro
    PRINT RIGHT$(a$, 5)    'Output is:  Basic
```

See Also MID$

LEN()

Returns the number of characters in a string or the number of bytes required to store a variable.

```
LEN(stringexpression$)
LEN(variable)
```

- _ stringexpression$ Any string expression.
- _ variable Any nonstring variable.

Example:

```
a$ = "Microsoft QBasic"
PRINT LEN(a$)
```

See Also OPEN

LTRIM$()
RTRIM$()

Remove leading and trailing spaces from a string.

```
LTRIM$(stringexpression$)
RTRIM$(stringexpression$)

    _ stringexpression$    Any string expression.

Example:
    a$ = "    Basic    "
    PRINT "*" + a$ + "*"            'Output is:  *    Basic    *
    PRINT "*" + LTRIM$(a$) + "*"    'Output is:  *Basic    *
    PRINT "*" + RTRIM$(a$) + "*"    'Output is:  *    Basic*
```

MID$()

The MID$ function returns part of a string (a substring).
The MID$ statement replaces part of a string variable with another string.

```
MID$(stringexpression$,start%[,length%])
MID$(stringvariable$,start%[,length%])=stringexpression$
```

_ stringexpression$	The string from which the MID$ function returns a substring, or the replacement string used by the MID$ statement. It can be any string expression.
_ start%	The position of the first character in the substring being returned or replaced.
_ length%	The number of characters in the substring. If the length is omitted, MID$ returns or replaces all characters to the right of the start position.
_ stringvariable$	The string variable being modified by the MID$ statement.

Example:

```
    a$ = "Where is Paris?"
    PRINT MID$(a$, 10, 5)       'Output is:  Paris
    Text$ = "Paris, France"
    PRINT Text$                 'Output is:  Paris, France
    MID$(Text$, 8) = "Texas "
    PRINT Text$                 'Output is:  Paris, Texas
```

See Also LEFT$, RIGHT$ LEN

SPACE$()

Returns a string of spaces.

```
SPACE$(n%)

    _ n%    The number of spaces you want in the string.

Example:
    FOR i% = 1 TO 5
        x$ = SPACE$ (i%)
        PRINT x$; i%
    NEXT i%

See Also     SPC      STRING$
```

STRING$()

Returns a string of a specified length made up of a repeating character.

```
STRING$(length%,{ascii-code% | stringexpression$})
```

_ length%	The length of the string.
_ ascii-code%	The ASCII code of the repeating character.
_ stringexpression$	Any string expression. STRING$ fills the string with the first character in stringexpression$.

Example:

```
PRINT STRING$(5, "-");
PRINT "Hello";
PRINT STRING$(5, "-")
```

See Also ASCII Character Codes

TIME$

```
The TIME$ function returns the computer's current system time.
The TIME$ statement sets the current system time on your computer.

TIME$
TIME$ = stringexpression$

    _ stringexpression$    The time in one of the following forms:
                             hh          Sets the hour; minutes and seconds
                                         default to 00.
                             hh:mm       Sets the hour and minutes; seconds
                                         default to 00.
                             hh:mm:ss    Sets the hour, minutes, and seconds.

    _ The TIME$ function returns a string in the form hh:mm:ss.

Example:
    PRINT TIME$
    TIME$ = "08:00:58"    'Note: The new system time remains in effect until
                          '      you change it again.
    PRINT "Time set to "; TIME$

See Also     DATE$
```

9 Boolean Operations

Content of section 9

Boolean Operators
NOT AND OR XOR EQV IMP

Boolean operators perform bit manipulations, Boolean operations, or tests on multiple relations. They return a true (nonzero) or false (zero) value to be used in making a decision.

```
result = expression1 boolean-operator expression2
```

_ boolean-operator Any of the following Boolean operators:

NOT	Bit-wise complement
AND	Conjunction
OR	Disjunction (inclusive "or")
XOR	Exclusive "or"
EQV	Equivalence
IMP	Implication

_ Each operator returns results as indicated in the following truth table. T is true (nonzero); F is false (zero):

Expression1	Expression2	NOT	AND	OR	XOR	EQV	IMP
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	F	T
F	F	T	F	F	F	T	T

_ Boolean operations are performed after arithmetic and relational operations in order of precedence.
_ Expressions are converted to integers or long integers before a Boolean operation is performed.
_ If the expressions evaluate to 0 or -1, a Boolean operation returns 0 or -1 as the result. Because Boolean operators do bit-wise calculations, using values other than 0 for false and -1 for true may produce unexpected results.

10 Type Conversion Operations

Content of section 10

ASC()
CHR$()

```
ASC returns the ASCII code for the first character in a string expression.
CHR$ returns the character corresponding to a specified ASCII code.

ASC(stringexpression$)
CHR$(ascii-code%)

    _ stringexpression$    Any string expression.
    _ ascii-code%          The ASCII code of the desired character.

Example:
    PRINT ASC("Q")    'Output is:  81
    PRINT CHR$(65)    'Output is:  A

See Also    ASCII Character Codes
```

CDBL()
CSNG()

```
CDBL converts a numeric expression to a double-precision value.
CSNG converts a numeric expression to a single-precision value.

CDBL(numeric-expression)
CSNG(numeric-expression)

    _ numeric-expression    Any numeric expression.

Example:
    PRINT 1 / 3, CDBL(1 / 3)    'Output is:  .3333333  .3333333333333333
    PRINT CSNG(975.3421515#)    'Output is:  975.3422

See Also    CINT, CLNG
```

CINT()
CLNG()

CINT rounds a numeric expression to an integer.
CLNG rounds a numeric expression to a long (4-byte) integer.

```
CINT(numeric-expression)
CLNG(numeric-expression)
```

_ numeric-expression	For CINT, any numeric expression in the range -32,768 through 32,767. For CLNG, any numeric expression in the range -2,147,483,648 through 2,147,483,647.

Example:

```
    PRINT CINT(12.49), CINT(12.51)    'Output is:  12  13
    PRINT CLNG(338457.8)              'Output is:  338458
```

See Also CDBL, CSNG FIX, INT

MKI$() CVI()
MKL$() CVL()
MKS$() CVS()
MKD$() CVD()

MKI$, MKL$, MKS$, and MKD$ convert numbers to numeric strings that can be stored in FIELD statement string variables. CVI, CVL, CVS, and CVD convert those strings back to numbers.

```
MKI$(integer-expression%)
MKL$(long-integer-expression&)
MKS$(single-precision-expression!)
MKD$(double-precision-expression#)
CVI(2-byte-numeric-string)
CVL(4-byte-numeric-string)
CVS(4-byte-numeric-string)
CVD(8-byte-numeric-string)
```

Function	Returns	Function	Returns
MKI$	A 2-byte string	CVI	An integer
MKL$	A 4-byte string	CVL	A long integer
MKS$	A 4-byte string	CVS	A single-precision number
MKD$	An 8-byte string	CVD	A double-precision number

See Also FIELD MKSMBF$, MKDMBF$, CVSMBF, CVDMBF

MKSMBF$() CVSMBF ()
MKDMBF$() CVDMBF ()

MKSMBF$ and MKDMBF$ convert IEEE-format numbers to Microsoft-Binary-format numeric strings that can be stored in FIELD statement string variables. CVSMBF and CVDMBF convert those strings back to IEEE-format numbers.

```
MKSMBF$(single-precision-expression!)
MKDMBF$(double-precision-expression#)
CVSMBF (4-byte-numeric-string)
CVDMBF (8-byte-numeric-string)
```

Function	Returns
MKSMBF$	A 4-byte string containing a Microsoft-Binary-format number
MKDMBF$	An 8-byte string containing a Microsoft-Binary-format number
CVSMBF	A single-precision number in IEEE format
CVDMBF	A double-precision number in IEEE format

_ These functions are useful for maintaining data files created with older versions of Basic.

Example:

```
TYPE Buffer
    SngNum AS STRING * 4
    DblNum AS STRING * 8
END TYPE
DIM RecBuffer AS Buffer
OPEN "TESTDAT.DAT" FOR RANDOM AS #1 LEN = 12
SNum = 98.9
DNum = 645.3235622#
RecBuffer.SngNum = MKSMBF$(SNum)
RecBuffer.DblNum = MKDMBF$(DNum)
PUT #1, 1, RecBuffer
GET #1, 1, RecBuffer
CLOSE #1
PRINT CVSMBF(RecBuffer.SngNum), CVDMBF(RecBuffer.DblNum)
```

See Also FIELD MKn$, CVn

STR$()
VAL()

```
STR$ returns a string representation of a number.
VAL converts a string representation of a number to a number.

STR$(numeric-expression)
VAL(stringexpression$)

    _ numeric-expression    Any numeric expression.
    _ stringexpression$     A string representation of a number.

Example:
    PRINT "Decimal 65 is represented in hexadecimal as ";
    PRINT "&H" + LTRIM$(STR$(41))
    PRINT VAL(RIGHT$("Microsoft 1990", 4))
```

11 Error Trapping and Handling

Content of section 11

ERDEV
ERDEV$$

ERDEV returns an error code from the last device that generated a critical error. ERDEV$ returns the name of the device that generated the error.

ERDEV
ERDEV$

- The low byte of the value returned by ERDEV contains the DOS error code. The high byte contains device-attribute information.

Example

See Also ERR, ERL ON ERROR

ERR
ERL

ERR returns the run-time error code for the most recent error.
ERL returns the line number where the error occurred, or the closest line number before the line where the error occurred.

```
ERR
ERL
```

- ERL does not return line labels. If there are no line numbers in the program, ERL returns 0.

Example:

```
'Illustrates ERDEV, ERDEV$, ERL, ERR, ERROR, ON ERROR, and RESUME.
   ON ERROR GOTO Handler
10 CHDIR "a:\"                'Causes ERR 71 "Disk not ready"
                              'if no disk in Drive A.
20 y% = 0
30 x% = 5 / y%                'ERR 11 "Division by zero."
40 PRINT "x% ="; x%
50 ERROR 57                   'ERR 57 "Device I/O error."

Handler:
  PRINT
  PRINT "Error "; ERR; " on line "; ERL
  SELECT CASE ERR
    CASE 71
      PRINT "Using device "; ERDEV$; " device error code = "; ERDEV
      RESUME NEXT
    CASE 11
      INPUT "What value do you want to divide by"; y%
      RESUME                  'Retry line 30 with new value of y%.
    CASE ELSE
      PRINT "Unexpected error, ending program."
      END
  END SELECT
```

See Also ERDEV, ERDEV$ ERROR ON ERROR RESUME
Run-Time Error Codes

ERROR

Simulates an occurrence of a Basic error or a user-defined error.

```
ERROR expression%
```

_ expression%	The error code of a Basic or user-defined error; a value in the range 1 through 255. To define your own error, use a value that isn't listed in the Basic Run-Time Error Codes table.

Example:

```
'Illustrates ERDEV, ERDEV$, ERL, ERR, ERROR, ON ERROR, and RESUME.
   ON ERROR GOTO Handler
10 CHDIR "a:\"                'Causes ERR 71 "Disk not ready"
                              'if no disk in Drive A.
20 y% = 0
30 x% = 5 / y%                'ERR 11 "Division by zero."
40 PRINT "x% ="; x%
50 ERROR 57                   'ERR 57 "Device I/O error."

Handler:
  PRINT
  PRINT "Error "; ERR; " on line "; ERL
  SELECT CASE ERR
    CASE 71
      PRINT "Using device "; ERDEV$; " device error code = "; ERDEV
      RESUME NEXT
    CASE 11
      INPUT "What value do you want to divide by"; y%
      RESUME                  'Retry line 30 with new value of y%.
    CASE ELSE
      PRINT "Unexpected error, ending program."
      END
  END SELECT
```

See Also ERDEV, ERDEV$ ERROR ON ERROR RESUME
Run-Time Error Codes ERR, ERL

ON ERROR

Enables error handling and, when a run-time error occurs, directs your program to either branch to an error-handling routine or resume execution.

ON ERROR {GOTO line | RESUME NEXT}

- GOTO line — Branches to the first line of the error-handling routine, specified by a label or line number. To disable error handling, specify GOTO 0.
- RESUME NEXT — Resumes execution with the statement following the statement that caused the run-time error. Use the ERR function to obtain the error code for the error.

- If ON ERROR isn't used, any run-time error ends your program.

Example

See Also ERDEV, ERDEV$ ERR, ERL ERROR GOTO RESUME

RESUME

Resumes program execution after an error-handling routine.

```
RESUME [{line | NEXT}]
```

- _ line The label or number of the line where execution resumes. If line is 0 or omitted, execution resumes with the statement that caused the error.
- _ NEXT Resumes execution at the statement following the statement that caused the error.

Example

See Also ERROR ON ERROR

12 Event Trapping and Handling

Content of section 12

BEEP

Generates a beep sound from your computer's speaker.

BEEP

COM

COM enables, disables, or suspends event trapping on a communications port. If event trapping is enabled, ON COM branches to a subroutine whenever characters are received at the port.

```
COM(n%) ON
COM(n%) OFF
COM(n%) STOP
ON COM(n%) GOSUB line
```

_ n%	The number of a COM (serial) port (1 or 2).
_ COM(n%) ON	Enables trapping of a communications event.
_ COM(n%) OFF	Disables communications event trapping.
_ COM(n%) STOP	Suspends communications event trapping. Events are processed once event trapping is enabled by COM ON.
_ line	The label or number of the first line of the event-trapping subroutine.

Example:

```
    COM(1) ON       'Enable event trapping on port 1.
    ON COM(1) GOSUB ComHandler
    DO : LOOP WHILE INKEY$ = ""
    COM(1) OFF
    END

    ComHandler:
        PRINT "Something was typed at the terminal attached to COM1."
        RETURN
```

See Also OPEN COM

IOCTL
IOCTL$()

IOCTL transmits a control string to a device driver.
IOCTL$ returns current status information from a device driver.

```
IOCTL [#]filenumber%, string$
IOCTL$([#]filenumber%)
```

- filenumber% The number of an open device.
- string$ The control string sent to the device.

- IOCTL control strings and the information returned by IOCTL$ depend on the device driver. See your device-driver documentation for information about IOCTL control strings and what is returned by IOCTL$.

KEY()
ON KEY()

KEY enables, disables, or suspends event trapping of a key.
If event trapping is enabled, ON KEY branches to a subroutine whenever
the key is pressed.

```
KEY(n%) ON
KEY(n%) OFF
KEY(n%) STOP
ON KEY(n%) GOSUB line
```

_ n%	A value that specifies a function key, direction key, or user-defined key:

n%	Key
0	All keys listed here (KEY(0) ON, KEY(0) OFF, and KEY(0) STOP only).
1-10	Function keys F1-F10.
11	Up Arrow key.
12	Left Arrow key.
13	Right Arrow key.
14	Down Arrow key.
15-25	User-defined keys. For more information, see Declaring User-Defined Keys .
30, 31	Function keys F11 and F12.

_ KEY(n%) ON	Enables event trapping for the specified key.
_ KEY(n%) OFF	Disables key event trapping.
_ KEY(n%) STOP	Suspends key event trapping. Events are processed once event trapping is enabled by KEY ON.
_ line	The label or number of the first line of the event-trapping subroutine.

Example:

```
'This example requires Caps Lock and Num Lock to be off.
CONST ESC = 27
KEY 15, CHR$(&H4) + CHR$(&H1F)              'Set up Ctrl+S as KEY 15.
ON KEY(15) GOSUB PauseHandler
KEY(15) ON
WHILE INKEY$ <> CHR$(ESC)
    PRINT "Press Esc to stop, Ctrl+S to pause."
    PRINT
WEND
END

PauseHandler:
    SLEEP 1
    RETURN
```

See Also KEY (Assignment) Declaring User-Defined Keys

User Defined Keys

To declare a user-defined key, use the following variation of the KEY statement:

```
KEY n%, CHR$(keyboardflag%) + CHR$(scancode%)
```

- n% — A value in the range 15 through 25 that identifies the key.
- keyboardflag% — One of the following values, or a sum of values, specifying whether the user-defined key is used in combination with the Shift, Ctrl, Alt, NumLock, or Caps Lock keys, or with extended keys:

Value	Key
0	No keyboard flag
1 through 3	Either Shift key
4	Ctrl key
8	Alt key
32	NumLock key
64	Caps Lock key
128	Extended keys on a 101-key keyboard

  To specify multiple shift states, add the values together. For example, a value of 12 specifies that the user-defined key is used in combination with the Ctrl and Alt keys.

- scancode% — The scan code for the key being declared. See Keyboard Scan Codes .

PEN { ON | OFF | STOP }
ON PEN GOSUB ...

PEN enables, disables, or suspends light-pen event trapping.
If event trapping is enabled, ON PEN branches to a subroutine whenever
the light pen is activated.

```
PEN ON
PEN OFF
PEN STOP
ON PEN GOSUB line
```

_ PEN ON	Enables light-pen event trapping.
_ PEN OFF	Disables light-pen event trapping.
_ PEN STOP	Suspends light-pen event trapping. Events are processed once event trapping is enabled by PEN ON.
_ line	The label or number of the first line of the event-trapping subroutine.

Example:

```
    'This example requires a light pen.
    ON PEN GOSUB Handler
    PEN ON
    PRINT "Press Esc to exit."
    DO UNTIL INKEY$ = CHR$(27): LOOP
    END

Handler:
    PRINT "Pen is at row"; PEN(6); ", column"; PEN(7)
    RETURN
```

See Also PEN() Function

PEN()

Returns the status of the light pen.

PEN(n%)

_ n% Specifies the information to be returned about light pen status:

n%	Returns
0	Whether pen was used since last call (-1 if yes, 0 if no)
1	The x screen coordinate of the last pen press
2	The y screen coordinate of the last pen press
3	The current pen switch status (-1 if down, 0 if up)
4	The x screen coordinate where the pen last left the screen
5	The y screen coordinate where the pen last left the screen
6	The character row of the last pen press
7	The character column of the last pen press
8	The character row where the pen last left the screen
9	The character column where the pen last left the screen

Example:

```
DO
  P = PEN(3)
  LOCATE 1, 1: PRINT "Pen is ";
  IF P THEN PRINT "down" ELSE PRINT "up  "
  PRINT "X ="; PEN(4), " Y ="; PEN(5); "  "
LOOP
```

See Also PEN, ON PEN Statements SCREEN Screen Modes

PLAY { ON | OFF | STOP }
ON PLAY ... GOSUB ...

PLAY enables, disables, or suspends play event trapping.
If event trapping is enabled, ON PLAY branches to a subroutine whenever
the music buffer contains fewer than a specified number of notes.

```
PLAY ON
PLAY OFF
PLAY STOP
ON PLAY(queuelimit%) GOSUB line
```

_ PLAY ON	Enables play event trapping.
_ PLAY OFF	Disables play event trapping.
_ PLAY STOP	Suspends play event trapping. Events are processed once event trapping is enabled by PLAY ON.
_ queuelimit%	A number in the range 1 through 32. ON PLAY branches to a subroutine when there are fewer than queuelimit% notes in the music buffer.
_ line	The label or number of the first line of the event-trapping subroutine.

Example:

```
ON PLAY(3) GOSUB Background
PLAY ON
Music$ = "MBo3L8ED+ED+Eo2Bo3DCL2o2A"
PLAY Music$
LOCATE 2, 1: PRINT "Press any key to stop.";
DO WHILE INKEY$ = "": LOOP
END

Background:
    i% = i% + 1
    LOCATE 1, 1: PRINT "Background called "; i%; "time(s)";
    PLAY Music$
    RETURN
```

See Also PLAY (Music) PLAY() Function

PLAY()

Returns the number of notes in the background music queue.

PLAY(n)

    _ n   Any numeric expression.

Example:

```
    Music$ = "MBT180o2P2P8L8GGGL2E-P24P8L8FFFL2D"
    PLAY Music$
    WHILE PLAY(0) > 5: WEND
    PRINT "Just about done!"
```

See Also PLAY (Music) PLAY, ON PLAY (Event Trapping)

SLEEP

Suspends program execution.

```
SLEEP [seconds&]
```

- seconds& Number of seconds to suspend the program.

- If seconds& is 0 or is omitted, the program is suspended until a key is pressed or a trapped event occurs.

Example:

```
PRINT "Taking a 10-second nap..."
SLEEP 10
PRINT "Wake up!"
```

See Also WAIT

SOUND

Generates a sound through your computer's speaker.

SOUND frequency, duration

- _ frequency — The frequency of the sound in hertz; a value in the range 37 through 32,767.
- _ duration — The number of system clock ticks the sound lasts; a value in the range 0 through 65,535. There are 18.2 clock ticks per second.

Example:

```
FOR i% = 440 TO 1000 STEP 5
SOUND i%, i% / 1000
NEXT i%
```

See Also PLAY

STICK()

Returns the coordinates of a joystick.

STICK(n%)

_ n% Indicates the coordinate returned:

n%	Returns
0	x coordinate of joystick A
1	y coordinate of joystick A
2	x coordinate of joystick B
3	y coordinate of joystick B

_ You must call STICK(0) before STICK(1), STICK (2), or STICK(3). STICK(0) records the current coordinates.

Example:

```
Temp% = STICK(0)
PRINT STICK(2), STICK(3)
```

See Also STRIG Function STRIG, ON STRIG Statements

STRIG() { ON | OFF | STOP }
ON STRIG() GOSUB ...

STRIG enables, disables, or suspends joystick event trapping.
If event trapping is enabled, ON STRIG branches to a subroutine whenever
a specified joystick trigger is pressed.

```
STRIG(n%) ON
STRIG(n%) OFF
STRIG(n%) STOP
ON STRIG(n%) GOSUB line
```

_ n% A value that specifies a joystick trigger:

n%	Trigger
0	Lower trigger, joystick A
2	Lower trigger, joystick B
4	Upper trigger, joystick A
6	Upper trigger, joystick B

_ STRIG(n%) ON Enables joystick event trapping.
_ STRIG(n%) OFF Disables joystick event trapping.
_ STRIG(n%) STOP Suspends joystick event trapping. Events are processed once event trapping is enabled by STRIG ON.
_ line The label or number of the first line of the event-trapping subroutine.

Example:

```
    'This example requires a joystick.
    ON STRIG(0) GOSUB Handler
    STRIG(0) ON
    PRINT "Press Esc to exit."
    DO UNTIL INKEY$ = CHR$(27): LOOP
    END

Handler:
    PRINT "Joystick trigger is depressed."
    RETURN
```

See Also STICK STRIG() Function

STRIG()

Returns the status of a joystick trigger.

STRIG(n%)

_ n% A value that specifies a joystick status condition:

n%	Condition
0	Lower joystick A trigger was pressed since last STRIG(0)
1	Lower joystick A trigger is currently pressed
2	Lower joystick B trigger was pressed since last STRIG(2)
3	Lower joystick B trigger is currently pressed
4	Upper joystick A trigger was pressed since last STRIG(4)
5	Upper joystick A trigger is currently pressed
6	Upper joystick B trigger was pressed since last STRIG(6)
7	Upper joystick B trigger is currently pressed

_ STRIG returns -1 if the condition is true, 0 otherwise.

Example:

```
PRINT "Press Esc to exit."
DO
    IF STRIG(0) OR INKEY$ = CHR$(27) THEN EXIT DO
LOOP
DO
    BEEP                  'BEEP while trigger A is pressed.
LOOP WHILE STRIG(1)
```

See Also STICK STRIG, ON STRIG Statements

TIMER { ON | OFF | STOP }
ON TIMER() GOSUB ...

TIMER enables, disables, or suspends timer event trapping.
If event trapping is enabled, ON TIMER branches to a subroutine whenever
a specified number of seconds has elapsed.

```
TIMER ON
TIMER OFF
TIMER STOP
ON TIMER(n%) GOSUB line
```

_ TIMER ON	Enables timer event trapping.
_ TIMER OFF	Disables timer event trapping.
_ TIMER STOP	Suspends timer event trapping. Events are processed once event trapping is enabled by TIMER ON.
_ n%	The number of seconds that elapse before ON TIMER branches to the event-trapping subroutine; a value in the range 1 through 86,400 (24 hours).
_ line	The label or number of the first line of the event-trapping subroutine.

Example:

```
ON TIMER(1) GOSUB TimeUpdate
TIMER ON
CLS
PRINT "Time: "; TIME$
StartTime = TIMER
WHILE TimePast < 10
    TimePast = TIMER - StartTime
WEND
END

TimeUpdate:
    LOCATE 1, 7: PRINT TIME$
    RETURN
```

See Also TIMER() Function

VARPTR$()

Returns a string representation of a variable's address for use in DRAW and PLAY statements.

```
VARPTR$(commandstring$)
```

_ commandstring$ A string variable containing DRAW or PLAY commands.

Example:

```
Scale$ = "CDEFGAB"
PLAY "L16"
FOR i% = 0 TO 6
    PLAY "O" + STR$(i%)
    PLAY "X" + VARPTR$(Scale$)
NEXT i%
```

See Also DRAW PLAY (Music) VARPTR, VARSEG

13 Low Level and Memory Management

Content of section 13

BSAVE
BLOAD

BSAVE copies the contents of an area of memory to a file.
BLOAD loads a file created by BSAVE into memory.

```
BSAVE filespec$, offset%, length&
BLOAD filespec$[,offset%]
```

- filespec$ — For BSAVE, a file to which an area of memory (a byte-for-byte memory image) is copied. For BLOAD, a memory-image file created by a previous BSAVE.
- offset% — For BSAVE, the offset of the starting address of the area of memory being saved. For BLOAD, the offset of the address where loading starts.
- length& — The number of bytes to copy (from 0 through 65,535).

- The starting address of the memory area saved or loaded is determined by the offset and the most recent DEF SEG statement.

See Also DEF SEG VARPTR, VARSEG

CALL ABSOLUTE()

Transfers control to a machine-language procedure.

```
CALL ABSOLUTE ([argumentlist,] offset%)

    _ argumentlist    Arguments passed to a machine-language procedure
                      as offsets from the current data segment.
    _ offset%         The offset from the current code segment, set by
                      DEF SEG, to the starting location of the procedure.
```

Example:

```
    'Calls routine for printing the screen to a local printer.
    DIM a%(2)
    DEF SEG = VARSEG(a%(0))
    FOR i% = 0 TO 2
       READ d%
       POKE VARPTR(a%(0)) + i%, d%
    NEXT i%
    DATA 205, 5, 203  : ' int 5  retf  'Machine-language code
                                       'for printing screen.
    CALL ABSOLUTE(VARPTR(a%(0)))
    DEF SEG
```

See Also CALL VARPTR, VARSEG

CLEAR

Closes all files, releases file buffers, clears all common variables, sets numeric variables and arrays to zero, sets string variables to null, and initializes the stack. Optionally, CLEAR also changes the size of the stack.

```
CLEAR [,,stack&]
```

    _ stack&   Sets the size (in bytes) of stack space for your program.

Example:

```
    CLEAR ,,2000
```

See Also   ERASE

DEF SEG

Sets the current segment address.

```
DEF SEG [=address]

    _ address    A segment address used by BLOAD, BSAVE, CALL ABSOLUTE,
                 PEEK, or POKE; a value in the range 0 through 65,535. If
                 address is omitted, DEF SEG resets the current segment
                 address to the default data segment.

Example:
    DEF SEG = 0
    Status% = PEEK(&H417)              'Read keyboard status.
    POKE &H417, (Status% XOR &H40)     'Change Caps Lock state, bit 6.

See Also    BSAVE, BLOAD    CALL ABSOLUTE    PEEK, POKE
```

FRE()

Returns the amount (in bytes) of available or unused memory.

```
FRE(numeric-expression)
FRE(stringexpression$)
```

_ numeric-expression — A value that specifies the type of memory:

Value	FRE returns
-1	The size of the largest array (nonstring) you can create
-2	The unused stack space
Any other number	The available string space

_ stringexpression$ — Any string expression. FRE compacts the free string space into a single block, then returns the amount of available string space.

Example:

```
PRINT "String Space", FRE("")
PRINT "Unused Stack Space", FRE(-2)
PRINT "Array Space", FRE(-1)
```

PEEK()
POKE

```
PEEK returns a byte value stored at a specified memory location.
POKE writes a byte value to a specified memory location.

PEEK(address)
POKE address,byte%

    _ address    A byte position relative to the current segment address
                 set by DEF SEG; a value in the range 0 through 65,535.
    _ byte%      A byte value to write to the specified memory location;
                 a value in the range 0 through 255.

Example:
    DEF SEG = 0
    Status% = PEEK(&H417)              'Read keyboard status.
    POKE &H417, (Status% XOR &H40)     'Change Caps Lock state, bit 6.

See Also     DEF SEG
```

VARPTR()
VARSEG()

VARPTR returns the offset address of a variable.
VARSEG returns the segment address of a variable.

VARPTR(variablename)
VARSEG(variablename)

_ variablename Any Basic variable.

See Also CALL ABSOLUTE DEF SEG PEEK, POKE VARPTR$

VARPTR$()

Returns a string representation of a variable's address for use in DRAW and PLAY statements.

VARPTR$(commandstring$)

_ commandstring$ A string variable containing DRAW or PLAY commands.

Example:

```
Scale$ = "CDEFGAB"
PLAY "L16"
FOR i% = 0 TO 6
    PLAY "O" + STR$(i%)
    PLAY "X" + VARPTR$(Scale$)
NEXT i%
```

See Also DRAW PLAY (Music) VARPTR, VARSEG

14 DOS Shell Commands

Content of section 14

CHDIR
MKDIR
RMDIR
FILES

CHDIR changes a drive's default directory. MKDIR creates a subdirectory. MRDIR removes a subdirectory. FILES displays the contents of the current directory or a specified directory.

```
CHDIR pathname$
MKDIR pathname$
RMDIR pathname$
FILES [filespec$]
```

- _ pathname$ — The path of the new default directory, subdirectory to create, or subdirectory to remove.
- _ filespec$ — A filename or path (may include a drive and DOS wildcard characters). If you don't specify a filespec$, FILES displays all files in the current directory.

Example:

```
MKDIR "C:\TEMP\TEST"
CHDIR "C:\TEMP"
FILES
RMDIR "TEST"
```

KILL

Deletes files from disk.

```
KILL filespec$
```

```
    _ filespec$    Identifies the file or files to delete. It may include
                   a path and the DOS wildcard characters ? and *.
```

Example:

```
    INPUT "File to delete: "; f$
    KILL f$
```

See Also FILES

NAME

Renames a file or directory.

```
NAME oldspec$ AS newspec$
```

_ oldspec$ and newspec$	The name of an existing file and the new name for the file. Each name may include a path.

Example:

```
INPUT "Old Name: "; OldFN$
INPUT "New Name: "; NewFN$
NAME OldFN$ AS NewFN$
```

See Also FILES

ENVIRON$()
ENVIRON

ENVIRON$ returns a DOS environment string.
ENVIRON changes or adds an environment string in the DOS environment table.

```
ENVIRON$ (env-variable$)
ENVIRON$ (n%)
ENVIRON stringexpression$
```

_ env-variable$	The name of a DOS environment variable.
_ n%	Specifies that ENVIRON$ returns the nth string from the environment string table.
_ stringexpression$	The name and setting of a DOS environment variable (such as PATH or PROMPT) in one of the following forms: env-variable$=env-string$ env-variable$ env-string$

_ Changes made by the ENVIRON statement are erased when the program ends.

Example:

```
ENVIRON "PATH=TEST"
PRINT ENVIRON$("PATH")
```

SHELL

Suspends execution of a Basic program to run a DOS command or batch file.

SHELL [commandstring$]

- commandstring$ The name of a DOS command or batch file.

- Your program resumes when the DOS command or batch file completes.
- If you omit the command string, SHELL invokes a DOS shell and displays the DOS prompt. Use the EXIT command to resume your program.

Example:

```
SHELL "DIR"
```

15 Miscellaneous

Content of section 15

REM
'

Allows explanatory remarks to be inserted in a program.

```
REM remark
' remark
```

    _ remark   Any text.

    _ Remarks are ignored when the program runs unless they contain
      metacommands.
    _ A remark can be inserted on a line after an executable statement if it
      is preceded by the single-quote (') form of REM or if REM is preceded
      by a colon (:).

Example:

```
    REM    This is a comment.
    '      This is also a comment.
    PRINT "Test1"       'This is a comment after a PRINT statement.
    PRINT "Test2" :  REM This is also a comment after a PRINT statement.
```

See Also $STATIC, $DYNAMIC

TRON
TROFF

TRON and TROFF enable and disable tracing of program statements.

```
TRON
TROFF
```

    _ QBasic's debugging features make these statements unnecessary.

See Also Run and Debug Keys

Appendices

Contents of Appendices

Appendix A ASCII Character Set (DOS)

Regular ASCII Chart (character codes 0 - 127)

000	(nul)	016 ▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Extended ASCII Chart (character codes 128 - 255)

128	Ç	143	Å	158	×	172	¼	186	‖	200	╚	214	╓	228	õ	242	≥
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	╫	229	Õ	243	≤
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	╪	230	µ	244	⌠
131	â	146	Æ	161	í	175	»	189	¢	203	╦	217	┘	231	þ	245	⌡
132	ä	147	ô	162	ó	176	░	190	¥	204	╠	218	┌	232	Þ	246	÷
133	à	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Ú	247	≈
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Û	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	¤	221	▌	235	Ù	249	∙
136	ê	151	ù	166	ª	180	┤	194	┬	208	ð	222	▐	236	ý	250	·
137	ë	152	ÿ	167	º	181	Á	195	├	209	Ð	223	▀	237	Ý	251	√
138	è	153	Ö	168	¿	182	Â	196	─	210	Ê	224	Ó	238	¯	252	ⁿ
139	ï	154	Ü	169	⌐	183	À	197	┼	211	Ë	225	ß	239	´	253	²
140	î	155	ø	170	¬	184	©	198	ã	212	È	226	Ô	240	-	254	■
141	ì	156	£	171	½	185	╣	199	Ã	213	ı	227	Ò	241	±	255	
142	Ä	157	Ø												

Appendix B Keyboard Scan Codes

Key	Code	Key	Code	Key	Code
Esc	1	A	30	Caps Lock	58
! or 1	2	S	31	F1	59
@ or 2	3	D	32	F2	60
# or 3	4	F	33	F3	61
$ or 4	5	G	34	F4	62
% or 5	6	H	35	F5	63
^ or 6	7	J	36	F6	64
& or 7	8	K	37	F7	65
* or 8	9	L	38	F8	66
(or 9	10	: or ;	39	F9	67
) or 0	11	" or '	40	F10	68
_ or -	12	~ or `	41	F11	133
+ or =	13	Left Shift	42	F12	134
Bksp	14	| or \	43	NumLock	69
Tab	15	Z	44	Scroll Lock	70
Q	16	X	45	Home or 7	71
W	17	C	46	Up or 8	72
E	18	V	47	PgUp or 9	73
R	19	B	48	Gray -	74
T	20	N	49	Left or 4	75
Y	21	M	50	Center or 5	76
U	22	< or ,	51	Right or 6	77
I	23	> or .	52	Gray +	78
O	24	? or /	53	End or 1	79
P	25	Right Shift	54	Down or 2	80
{ or [	26	Prt Sc or *	55	PgDn or 3	81
} or]	27	Alt	56	Ins or 0	82
Enter	28	Spacebar	57	Del or .	83
Ctrl	29				

Appendix C Colour Attributes & Values

	Color monitor		Monochrome monitor	
Color attribute	Default color value(a)	Displayed color	Default color value	Displayed color
SCREEN Modes 0, 7, 8, 9(b), 12, and 13				
0	0	Black	0(c)	Off
1	1	Blue		Underlined(d)
2	2	Green	1(c)	On(d)
3	3	Cyan	1(c)	On(d)
4	4	Red	1(c)	On(d)
5	5	Magenta	1(c)	On(d)
6	6	Brown	1(c)	On(d)
7	7	White	1(c)	On(d)
8	8	Gray	0(c)	Off
9	9	Light Blue		High-intensity Underlined
10	10	Light green	2(c)	High-intensity
11	11	Light cyan	2(c)	High-intensity
12	12	Light red	2(c)	High-intensity
13	13	Light magenta	2(c)	High-intensity
14	14	Yellow	2(c)	High-intensity
15	15	High-intensity white	0(c)	Off
SCREEN Modes 1 and 9(e)				
0	0	Black	0	Off
1	11	Light cyan	2	High-intensity
2	13	Light magenta	2	High-intensity
3	15	High-intensity white	0	Off white
SCREEN Modes 2 and 11				
0	0	Black	0	Off
1	15	High-intensity white	0	Off white

(a) EGA color numbers. VGA and MCGA use display-color values that produce visually equivalent colors.
(b) For VGA or EGA with video memory > 64K.
(c) Only for mode 0.
(d) Off when used for background.
(e) EGA with video memory <= 64K.

See Also COLOR PALETTE, PALETTE USING SCREEN
Screen Modes

Appendix D Screen Modes

The following table summarizes screen modes:

```
----------MDPA, CGA, Hercules, Olivetti, EGA, VGA, or MCGA Adapters-----------
SCREEN 0: Text mode only
    _ 40 x 25, 40 x 43, 40 x 50, 80 x 25, 80 x 43, or 80 x 50 text format,
      8 x 8 character box (8 x 14, 9 x 14, or 9 x 16 with EGA or VGA)
    _ 16 colors assigned to any of 16 attributes (with CGA or EGA)
    _ 64 colors assigned to any of 16 attributes (with EGA or VGA)
    _ Depending on the text resolution and adapter, 8 video memory pages
      (0-7), 4 pages (0-3), 2 pages (0-1), or 1 page (0)

----------------------CGA, EGA, VGA, or MCGA Adapters-----------------------
SCREEN 1: 320 x 200 graphics
    _ 40 x 25 text format, 8 x 8 character box
    _ 16 background colors and one of two sets of 3 foreground colors
      assigned using COLOR statement with CGA
    _ 16 colors assigned to 4 attributes with EGA or VGA
    _ 1 video memory page (0)
SCREEN 2: 640 x 200 graphics
    _ 80 x 25 text format, 8 x 8 character box
    _ 16 colors assigned to 2 attributes with EGA or VGA
    _ 1 video memory page (0)

---------------------Hercules, Olivetti, or AT&T Adapters--------------------
SCREEN 3: Hercules adapter required, monochrome monitor only
    _ 720 x 348 graphics
    _ 80 x 25 text format, 9 x 14 character box
    _ Usually 2 video memory pages (0-1); 1 page (0) if a second color
      display adapter is installed
    _ PALETTE statement not supported
    _ Invoke the Hercules driver MSHERC.COM before using screen mode 3
SCREEN 4:
    _ Supports Olivetti Personal Computers models M24, M240, M28, M280,
      M380, M380/C, and M380/T and AT&T Personal Computers 6300 series
    _ 640 x 400 graphics
    _ 80 x 25 text format, 8 x 16 character box
    _ 1 of 16 colors assigned as the foreground color (selected by the
      COLOR statement); background is fixed at black
    _ 1 video memory page (0)
    _ PALETTE statement not supported

----------------------------EGA or VGA Adapters-----------------------------
SCREEN 7: 320 x 200 graphics
    _ 40 x 25 text format, 8 x 8 character box
    _ Assignment of 16 colors to any of 16 attributes
    _ If 64K EGA adapter memory, 2 video memory pages (0-1); otherwise,
      8 pages (0-7)
SCREEN 8: 640 x 200 graphics
    _ 80 x 25 text format, 8 x 8 character box
    _ Assignment of 16 colors to any of 16 attributes
    _ If 64K EGA adapter memory, 1 video memory page (0); otherwise,
      4 pages (0-3)
SCREEN 9: 640 x 350 graphics
    _ 80 x 25 or 80 x 43 text format, 8 x 14 or 8 x 8 character box
    _ 16 colors assigned to 4 attributes (64K adapter memory), or
      64 colors assigned to 16 attributes (more than 64K adapter memory)
    _ If 64K EGA adapter memory, 1 video memory page (0); otherwise,
      2 pages (0-1)
```

```
-----------------EGA or VGA Adapters, Monochrome Monitor Only----------------
SCREEN 10: 640 x 350 graphics, monochrome monitor only
    _ 80 x 25 or 80 x 43 text format, 8 x 14 or 8 x 8 character box
    _ Up to 9 pseudocolors assigned to 4 attributes
    _ 2 video memory pages (0-1), 256K adapter memory required

----------------------------VGA or MCGA Adapters----------------------------
Screen 11 (VGA or MCGA)
    _ 640 x 480 graphics
    _ 80 x 30 or 80 x 60 text format, 8 x 16 or 8 x 8 character box
    _ Assignment of up to 256K colors to 2 attributes
    _ 1 video memory page (0)
Screen 12 (VGA)
    _ 640 x 480 graphics
    _ 80 x 30 or 80 x 60 text format, 8 x 16 or 8 x 8 character box
    _ Assignment of up to 256K colors to 16 attributes
    _ 1 video memory page (0)
Screen 13 (VGA or MCGA)
    _ 320 x 200 graphics
    _ 40 x 25 text format, 8 x 8 character box
    _ Assignment of up to 256K colors to 256 attributes
    _ 1 video memory page (0)
```

Appendix D Run-Time Error Codes

Code	Message	Code	Message
1	NEXT without FOR	37	Argument-count mismatch
2	Syntax error	38	Array not defined
3	RETURN without GOSUB	40	Variable required
4	Out of DATA	50	FIELD overflow
5	Illegal function call	51	Internal error
6	Overflow	52	Bad file name or number
7	Out of memory	53	File not found
8	Label not defined	54	Bad file mode
9	Subscript out of range	55	File already open
10	Duplicate definition	56	FIELD statement active
11	Division by zero	57	Device I/O error
12	Illegal in direct mode	58	File already exists
13	Type mismatch	59	Bad record length
14	Out of string space	61	Disk full
16	String formula too complex	62	Input past end of file
17	Cannot continue	63	Bad record number
18	Function not defined	64	Bad file name
19	No RESUME	67	Too many files
20	RESUME without error	68	Device unavailable
24	Device timeout	69	Communication-buffer overflow
25	Device fault	70	Permission denied
26	FOR without NEXT	71	Disk not ready
27	Out of paper	72	Disk-media error
29	WHILE without WEND	73	Feature unavailable
30	WEND without WHILE	74	Rename across disks
33	Duplicate label	75	Path/File access error
35	Subprogram not defined	76	Path not found

Appendix E Example Code - REMLINE.BAS

REMLINE.BAS - This is the example code supplied with QBASIC V1.1. The program strips the line numbers from files created for use with BASICA (GWBASIC), in which line numbers are compulsory.

The file has been formatted for readability in this document.

```
<START OF FILE>

DEFINT A-Z
'
'   Microsoft RemLine - Line Number Removal Utility
'   Copyright (C) Microsoft Corporation   - 1985, 1986, 1987
'
'   REMLINE.BAS is a program to remove line numbers from Microsoft BASIC
'   Programs. It removes only those line numbers that are not the object
'   of one of the following statements: GOSUB, RETURN, GOTO, THEN, ELSE,
'   RESUME, RESTORE, or RUN.
'
'   REMLINE is run by typing
'
'      REMLINE [<input> [, <output>]]
'
'   where <input> is the name of the file to be processed and <output>
'   is the name of the file or device to receive the reformatted output.
'   If no extension is given, .BAS is assumed (except for output devices).
'   If file names are not given, REMLINE prompts for file names. If both
'   file names are the same, REMLINE saves the original file with the
'   extension .BAK.
'
'   REMLINE makes several assumptions about the program:
'
'     1. It must be correct syntactically, and must run in BASICA or
'        GWBASIC interpreter.
'     2. There is a 400 line limit. To process larger files, change
'        MaxLines constant.
'     3. The first number encountered on a line is considered a line
'        number; thus some continuation lines (in a compiler specific
'        constructiion) may not be handled correctly.
'     4. REMLINE can handle simple statements that test the ERL function
'        using  relational operators such as =, <, and >. For example,
'        the following statement is handled correctly:
'
'             IF ERL = 100 THEN END
'
'        Line 100 is not removed from the source code. However, more
'        complex expressions that contain the +, -, AND, OR, XOR, EQV,
'        MOD, or IMP operators may not be handled correctly. For example,
'        in the following statement REMLINE does not recognize line 105
'        as a referenced line number and removes it from the source code:
'
'             IF ERL + 5 = 105 THEN END
'
'   If you do not like the way REMLINE formats its output, you can modify
'   the output lines in SUB GenOutFile. An example is shown in comments.
```

```
' Function and Subprogram declarations

DECLARE FUNCTION GetToken$ (Search$, Delim$)
DECLARE FUNCTION StrSpn% (InString$, Separator$)
DECLARE FUNCTION StrBrk% (InString$, Separator$)
DECLARE FUNCTION IsDigit% (Char$)
DECLARE SUB GetFileNames ()
DECLARE SUB BuildTable ()
DECLARE SUB GenOutFile ()
DECLARE SUB InitKeyTable ()

' Global and constant data

CONST TRUE = -1
CONST false = 0
CONST MaxLines = 400

DIM SHARED LineTable!(MaxLines)
DIM SHARED LineCount
DIM SHARED Seps$, InputFile$, OutputFile$, TmpFile$

' Keyword search data

CONST KeyWordCount = 9
DIM SHARED KeyWordTable$(KeyWordCount)

KeyData:
   DATA THEN, ELSE, GOSUB, GOTO, RESUME, RETURN, RESTORE, RUN, ERL, ""

' Start of module-level program code

   Seps$ = " ,:=<>()" + CHR$(9)
   InitKeyTable
   GetFileNames
   ON ERROR GOTO FileErr1
   OPEN InputFile$ FOR INPUT AS 1
   ON ERROR GOTO 0
   COLOR 7: PRINT "Working"; : COLOR 23: PRINT " . . .": COLOR 7: PRINT
   BuildTable
   CLOSE #1
   OPEN InputFile$ FOR INPUT AS 1
   ON ERROR GOTO FileErr2
   OPEN OutputFile$ FOR OUTPUT AS 2
   ON ERROR GOTO 0
   GenOutFile
   CLOSE #1, #2
   IF OutputFile$ <> "CON" THEN CLS

END
```

```
FileErr1:
   CLS
   PRINT "      Invalid file name": PRINT
   INPUT "      New input file name (ENTER to terminate): ", InputFile$
   IF InputFile$ = "" THEN END
FileErr2:
   INPUT "      Output file name (ENTER to print to screen) :", OutputFile$
   PRINT
   IF (OutputFile$ = "") THEN OutputFile$ = "CON"
   IF TmpFile$ = "" THEN
      RESUME
   ELSE
      TmpFile$ = ""
      RESUME NEXT
   END IF

'
' BuildTable:
'   Examines the entire text file looking for line numbers that are
'   the object of GOTO, GOSUB, etc. As each is found, it is entered
'   into a table of line numbers. The table is used during a second
'   pass (see GenOutFile), when all line numbers not in the list
'   are removed.
' Input:
'   Uses globals KeyWordTable$, KeyWordCount, and Seps$
' Output:
'   Modefies LineTable! and LineCount
'
SUB BuildTable STATIC

   DO WHILE NOT EOF(1)
      ' Get line and first token
      LINE INPUT #1, InLin$
      token$ = GetToken$(InLin$, Seps$)
      DO WHILE (token$ <> "")
         FOR KeyIndex = 1 TO KeyWordCount
            ' See if token is keyword
            IF (KeyWordTable$(KeyIndex) = UCASE$(token$)) THEN
               ' Get possible line number after keyword
               token$ = GetToken$("", Seps$)
               ' Check each token to see if it is a line number
               ' (the LOOP is necessary for the multiple numbers
               ' of ON GOSUB or ON GOTO). A non-numeric token will
               ' terminate search.
               DO WHILE (IsDigit(LEFT$(token$, 1)))
                  LineCount = LineCount + 1
                  LineTable!(LineCount) = VAL(token$)
                  token$ = GetToken$("", Seps$)
                  IF token$ <> "" THEN KeyIndex = 0
               LOOP
            END IF
         NEXT KeyIndex
         ' Get next token
         token$ = GetToken$("", Seps$)
      LOOP
   LOOP

END SUB
```

```
'
' GenOutFile:
'  Generates an output file with unreferenced line numbers removed.
' Input:
'  Uses globals LineTable!, LineCount, and Seps$
' Output:
'  Processed file
'
SUB GenOutFile STATIC

   ' Speed up by eliminating comma and colon (can't separate first token)
   Sep$ = " " + CHR$(9)
   DO WHILE NOT EOF(1)
      LINE INPUT #1, InLin$
      IF (InLin$ <> "") THEN
         ' Get first token and process if it is a line number
         token$ = GetToken$(InLin$, Sep$)
         IF IsDigit(LEFT$(token$, 1)) THEN
            LineNumber! = VAL(token$)
            FoundNumber = false
            ' See if line number is in table of referenced line numbers
            FOR index = 1 TO LineCount
               IF (LineNumber! = LineTable!(index)) THEN
                  FoundNumber = TRUE
               END IF
            NEXT index
            ' Modify line strings
            IF (NOT FoundNumber) THEN
               token$ = SPACE$(LEN(token$))
               MID$(InLin$, StrSpn(InLin$, Sep$), LEN(token$)) = token$
            END IF

            ' You can replace the previous lines with your own
            ' code to reformat output. For example, try these lines:

            'TmpPos1 = StrSpn(InLin$, Sep$) + LEN(Token$)
            'TmpPos2 = TmpPos1 + StrSpn(MID$(InLin$, TmpPos1), Sep$)
            '
            'IF FoundNumber THEN
            '   InLin$ = LEFT$(InLin$, TmpPos1 - 1) + CHR$(9) + MID$(InLin$, TmpPos2)
            'ELSE
            '   InLin$ = CHR$(9) + MID$(InLin$, TmpPos2)
            'END IF

         END IF
      END IF
      ' Print line to file or console (PRINT is faster than console device)
      IF OutputFile$ = "CON" THEN
         PRINT InLin$
      ELSE
         PRINT #2, InLin$
      END IF
   LOOP

END SUB
```

```
'
' GetFileNames:
'  Gets a file name from COMMAND$ or by prompting the user.
' Input:
'  Used Command$ or user input
' Output:
'  Defines InputFiles$ and OutputFiles$
'
SUB GetFileNames STATIC

   IF (COMMAND$ = "") THEN
      CLS
      PRINT " Microsoft RemLine: Line Number Removal Utility"
      PRINT "       (.BAS assumed if no extension given)"
      PRINT
      INPUT "      Input file name (ENTER to terminate): ", InputFile$
      IF InputFile$ = "" THEN END
      INPUT "      Output file name (ENTER to print to screen): ", OutputFile$
      PRINT
      IF (OutputFile$ = "") THEN OutputFile$ = "CON"
   ELSE
      InputFile$ = UCASE$(GetToken$(COMMAND$, Seps$))
      OutputFile$ = UCASE$(GetToken$("", Seps$))
      IF (OutputFile$ = "") THEN
         INPUT "      Output file name (ENTER to print to screen): ", OutputFile$
         PRINT
         IF (OutputFile$ = "") THEN OutputFile$ = "CON"
      END IF
   END IF
   IF INSTR(InputFile$, ".") = 0 THEN
      InputFile$ = InputFile$ + ".BAS"
   END IF
   IF INSTR(OutputFile$, ".") = 0 THEN
      SELECT CASE OutputFile$
         CASE "CON", "SCRN", "PRN", "COM1", "COM2", "LPT1", "LPT2", "LPT3"
            EXIT SUB
         CASE ELSE
            OutputFile$ = OutputFile$ + ".BAS"
      END SELECT
   END IF
   DO WHILE InputFile$ = OutputFile$
      TmpFile$ = LEFT$(InputFile$, INSTR(InputFile$, ".")) + "BAK"
      ON ERROR GOTO FileErr1
      NAME InputFile$ AS TmpFile$
      ON ERROR GOTO 0
      IF TmpFile$ <> "" THEN InputFile$ = TmpFile$
   LOOP

END SUB
```

```
'
' GetToken$:
'  Extracts tokens from a string. A token is a word that is surrounded
'  by separators, such as spaces or commas. Tokens are extracted and
'  analyzed when parsing sentences or commands. To use the GetToken$
'  function, pass the string to be parsed on the first call, then pass
'  a null string on subsequent calls until the function returns a null
'  to indicate that the entire string has been parsed.
' Input:
'  Search$ = string to search
'  Delim$  = String of separators
' Output:
'  GetToken$ = next token
'
FUNCTION GetToken$ (Search$, Delim$) STATIC

   ' Note that SaveStr$ and BegPos must be static from call to call
   ' (other variables are only static for efficiency).
   ' If first call, make a copy of the string
   IF (Search$ <> "") THEN
      BegPos = 1
      SaveStr$ = Search$
   END IF

   ' Find the start of the next token
   NewPos = StrSpn(MID$(SaveStr$, BegPos, LEN(SaveStr$)), Delim$)
   IF NewPos THEN
      ' Set position to start of token
      BegPos = NewPos + BegPos - 1
   ELSE
      ' If no new token, quit and return null
      GetToken$ = ""
      EXIT FUNCTION
   END IF

   ' Find end of token
   NewPos = StrBrk(MID$(SaveStr$, BegPos, LEN(SaveStr$)), Delim$)
   IF NewPos THEN
      ' Set position to end of token
      NewPos = BegPos + NewPos - 1
   ELSE
      ' If no end of token, return set to end a value
      NewPos = LEN(SaveStr$) + 1
   END IF
   ' Cut token out of search string
   GetToken$ = MID$(SaveStr$, BegPos, NewPos - BegPos)
   ' Set new starting position
   BegPos = NewPos

END FUNCTION
```

```
'
' InitKeyTable:
'  Initializes a keyword table. Keywords must be recognized so that
'  line numbers can be distinguished from numeric constants.
' Input:
'  Uses KeyData
' Output:
'  Modifies global array KeyWordTable$
'
SUB InitKeyTable STATIC

   RESTORE KeyData
   FOR Count = 1 TO KeyWordCount
      READ KeyWord$
      KeyWordTable$(Count) = KeyWord$
   NEXT

END SUB

'
' IsDigit:
'  Returns true if character passed is a decimal digit. Since any
'  BASIC token starting with a digit is a number, the function only
'  needs to check the first digit. Doesn't check for negative numbers,
'  but that's not needed here.
' Input:
'  Char$ - initial character of string to check
' Output:
'  IsDigit - true if within 0 - 9
'
FUNCTION IsDigit (Char$) STATIC

   IF (Char$ = "") THEN
      IsDigit = false
   ELSE
      CharAsc = ASC(Char$)
      IsDigit = (CharAsc >= ASC("0")) AND (CharAsc <= ASC("9"))
   END IF

END FUNCTION
```

```
'
' StrBrk:
'  Searches InString$ to find the first character from among those in
'  Separator$. Returns the index of that character. This function can
'  be used to find the end of a token.
' Input:
'  InString$ = string to search
'  Separator$ = characters to search for
' Output:
'  StrBrk = index to first match in InString$ or 0 if none match
'
FUNCTION StrBrk (InString$, Separator$) STATIC

   Ln = LEN(InString$)
   BegPos = 1
   ' Look for end of token (first character that is a delimiter).
   DO WHILE INSTR(Separator$, MID$(InString$, BegPos, 1)) = 0
      IF BegPos > Ln THEN
         StrBrk = 0
         EXIT FUNCTION
      ELSE
         BegPos = BegPos + 1
      END IF
   LOOP
   StrBrk = BegPos

END FUNCTION

'
' StrSpn:
'  Searches InString$ to find the first character that is not one of
'  those in Separator$. Returns the index of that character. This
'  function can be used to find the start of a token.
' Input:
'  InString$ = string to search
'  Separator$ = characters to search for
' Output:
'  StrSpn = index to first nonmatch in InString$ or 0 if all match
'
FUNCTION StrSpn% (InString$, Separator$) STATIC

   Ln = LEN(InString$)
   BegPos = 1
   ' Look for start of a token (character that isn't a delimiter).
   DO WHILE INSTR(Separator$, MID$(InString$, BegPos, 1))
      IF BegPos > Ln THEN
         StrSpn = 0
         EXIT FUNCTION
      ELSE
         BegPos = BegPos + 1
      END IF
   LOOP
   StrSpn = BegPos

END FUNCTION

<END OF FILE>
```

Index

$

[

,

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X