QUICKCALC BASIC

Functions, Statements, and Commands

Notation used in this document:

expression	A BASIC expression. A string of variables, functions, and operations which evaluates to a number, string, or logic value.
num-expr	A numerical expression. Any expression which evaluates to a number (DOUBLE or LONGMATH) or a number string.
string-expr	A String expression. Any expression which evaluates to a string.
GUI mode	Refers to the Windows mode of running QuickCalc (see "Graphic User Interface mode vs. Console Mode).
console mode	Refers to the "DOS window" or "command-line" mode of running QuickCalc (see "QuickCalc User Interface).
stmt-num	A statement number. An unsigned integer used to number a line in the program.
value	A number string, that is, a character string that represents a number, not enclosed in quotes.
file-num	A file number, or an expression that evaluates to a file number between 1 and the maximum number of files allowed (see "Customizing")
num-var	A numeric variable, DOUBLE or LONGMATH. (Not an expression).
string-var	A string variable.
clause	One or more BASIC statements, separated by colons (:).
arg	An argument, or value passed to a function. An argument can usually be an expession which evaluates to a number or string as required by the function.
new	This feature is new to this version of BASIC. Not necessarily part of the "standard" BASIC.
file-spec	A file specification. A string which identifies a file. This may be the file name or a path \ name.
string-array	A special notation for storing a LONGMATH number or hex string in a string array (see "Working With Long Numbers").

?	(statement)	Shorthand for **PRINT**.
??	(statement)	*(new)* Shorthand for **DEBUGPRINT**.
//	(statement)	*(new)* Shorthand for **REM** (similar to C-language).
@	(function)	@ (*string-expr*)

Indirect name reference *(new)*.
Substitutes the contents of *string-expr* as the name referenced in the statement. *string-expr* must contain a valid variable, statement or function **name**. The first characters not valid in a name and all subsequent characters will be ignored.
See "*Advanced Features in QuickCalc*" for more information.

ABS (function) **ABS** (*num-expr*)
Absolute value of numeric expression

ACOT (function) **ACOT** (*num-expr)*
Arccotangent of numeric expression. The angle is returned in radians, unless the **DEGREES** statement is given. (see **DEGREES** and **RADIANS**). (see also **ATN**).

ACOT (*x*) is valid for all [real] *x*, positive and negative. The result is an angle from $+\pi$ to 0 (or 180 to 0 degrees), which places the angle in the first or second quadrants. The arccotangent has no way of knowing if the angle is intended to be in the third or fourth quadrants.

Note: the arccotangent may also be calculated as $\pi/2 - \arctan(x)$, however, very large values of x will result in loss of precision as $\arctan(x)$ becomes very close to $\pi/2$.

APPEND (statement) **APPEND** *file-spec*
Appends the program specified by *file-spec* to the end of the currently loaded program. The original program and the appended program are concatenated in memory before the program is run.

file-spec (the file specification) must be a string. If the *file-spec* contains a colon, backslash(es) or embedded blanks, it must be in quotes – (see "Strings"). You may use any file specification that works. . If the path is in quotes, you must use a double backslash (\\) wherever a backslash is desired. (Don't use **..**\). If a complete path is not specified, the *file-spec* is appended to the **path from which the original program was loaded**.

This function is useful if you have a collection of subroutines that are used in many different programs. Simply group the subroutines together as a program and then **APPEND** them.
Note: The combination of the two programs must be syntactically correct. No duplicates (statement numbers, functions or array definitions). The two programs must fit in the space allocated to load programs as set in the "Customize" dialog.

Note: The **APPEND** statement may appear anywhere in the program. It does not get executed at run time. The appended code is always placed at the end.

Note: An appended program may append another program, if desired, or include sections of code (see **INCLUDE**). Be careful to avoid a recursive loop of **APPEND**s (program loads itself).

ASC (function) **ASC** (*string-expr*).
ASCII code for 1st char of string.
Note: **ASC** ("") = 0 (no error)

ATN (function) **ATN** (*num-expr*)
Arctangent of numeric expression. The angle is returned in radians, unless the **DEGREES** statement is given. (see **DEGREES** and **RADIANS**)
ATN (*x*) is valid for all [real] *x*, positive and negative. The result is an angle from - π /2 to + π /2 (or –90 to +90 degrees), which places the angle in the first or fourth quadrants. The arctangent has no way of knowing if the angle is intended to be in the second or third quadrants.

BEEP (statement) Make a sound. Plays the sound in the file "beep.wav", located in the directory from which QuickCalc is loaded. You can replace that file with a sound of your choosing, just call it beep.wav.

If you specify a .wav file as a parameter, *i.e.*, **BEEP** *filename*, The computer will play that wave file (if it can find it). If you specify a filename only, it will look in the current working directory. If you specify a complete path, it will look there. You can specify a string variable or a quoted string constant. If it is a constant, remember you must replace all back-slash characters with double back-slashes (\\ ).

Note: You can't use .mid or .mp3 files.

Although it is not intended for this purpose, it will play songs and other music if it is in .wav format. If a long wave file gets started and you want to stop it, enter **BEEP** "" or click the **Step** button if a program is running.

Note: You can find some interesting wave file sounds in the Windows\Media folder.

Note: In the "old" BASIC, you could create a beep by printing **CHR$**(7). This doesn't work in the GUI Windows environment.

BREAK (statement) **BREAK**
Breaks out of the current **FOR** or **WHILE** loop.
Causes an error message is the program is not currently in a loop.

BROWSECOMMPORTS$ (function) [GUI-mode only]

BROWSECOMMPORTS$ (*title-string*)

Displays a dialog box which allows you to select an active and available COMM port to use when **OPEN**-ing SERIAL files. Returns a string containing the name of the selected COMM port, *e.g.*, "COM1."

If the desired port is not connected or the external device is not on, the port will not appear in the list. Connect the device or power it on and then click the "Rescan" button. Cancelling this dialog will terminate the BASIC program.

You can use the resulting string in an **OPEN** statement, e.g.,

OPEN SERIAL, 2, **BROWSECOMMPORTS$** ("Select the Port"), ….

which will allow you to select the port at Open time.

BROWSEINPUTFILE$ (function) [GUI-mode only]

BROWSEINPUTFILE$ (*title-string*)

Displays a dialog box which allows you to browse through the files on your computer and select one to use as an input file.

Returns a string containing the full path to the selected file, which you can use in an **OPEN** statement, *e.g.,*

OPEN (**INPUT**, 2, **BROWSEINPUTFILE$** ("File to Read")

The *title-string* will be displayed at the top of the file selection dialog box. Use a title which will be meaningful to you when you run the program, so you know which file the program is requesting.

If you cancel out of the dialog box, you will get an error message and the BASIC program will terminate.

Note: **BROWSEINPUTFILE$** does not open the file.
Note: The file you specify must exist.

BROWSEOUTPUTFILE$ (function) [GUI-mode only]

BROWSEOUTPUTFILE$ (*title-string*)

Displays a dialog box which allows you to browse through the files on your computer and select one to use as an output file.

Returns a string containing the full path to the selected file, which you can use in an **OPEN** statement, *e.g.,*

OPEN (**OUTPUT**, 3, **BROWSEOUTPUTFILE$** ("File to Write")

The *title-string* will be displayed at the top of the file selection dialog box. Use a title which will be meaningful to you when you run the program, so you know which file the program is requesting.

If you cancel out of the dialog box, you will get an error message and the BASIC program will terminate.

Note: **BROWSEOUTPUTFILE$** does not open or create the file.
Note: If the file you specify already exists, it will be over-written when you open it. If not, it will be created at that time.
Note: When the "explorer-type" browse dialog is open, you may rename files, delete files, create new folders, etc.

CHAIN (statement) **CHAIN** *program-file-spec* [, **KEEPDATA**]

Causes the current program to be terminated and a new program to be run.

The new program is specified by the **string expression** *program-file-spec*.
The *program-file-spec* may be a program name or a full or partial path. If the complete path is not specified, the default path is the directory from which the **original program** was loaded.
Files which are currently open remain open and may be used in the chained program. If you want them closed, **CLOSE** them before doing the **CHAIN**.
Debug switches remain in effect.
If you are stepping through a program and encounter the **CHAIN** statement, you will continue stepping into the chained program.
If a graph is active, it remains active, and the chained program may plot more data onto it.
Because this statement is executed while the program is running, the program specification may be any string expression, calculated at run time or typed in by the user.
Any statements on the same line following the **CHAIN** statement will not be executed.

If the optional parameter **KEEPDATA** is specified, all program variables and arrays are not cleared, and their values may be used in the chained program. The **LONG** "**ALL**" switch remains in effect. Daylight Saving Time and Time Zone overrides remain in effect.

Other than the above, all program switches, etc., are reset, exactly as if the program were started with the **RUN** command.

Note: Be careful not to **CHAIN** a program to itself (directly or indirectly), as this could result in a never-ending program.

CHANGESHAPE (statement)

CHANGESHAPE *shape-id, parameters...*

Modifies the descriptive parameters of shapes that have already been drawn.

shape-id is a value retrieved from **LASTSHAPEID** immediately after a shape is drawn.

parameters are similar to the descriptive parameters used in the **SHAPE** statement when the shape is created. Not all parameters are valid for all shapes.

(See "*Advanced Graphics*" for more information about this statement including a list of parameters.

CHDIR (statement) **CHDIR** *string-expr*

Changes the current working directory to the path specified in *string-expr*. The current working directory is the default directory from which programs are loaded and files are referenced or created if a complete path is not specified.

If *string-expr* is a literal string, it must be enclosed in quotes, and if it contains back-slash characters, they must be replaced with double-backslashes, e.g. **"C:\\quickcalc\\programs"**. If the string does not specify a complete path, it will look for it in the current working directory.

Console Mode: The path must already exist. The current working directory is changed to the new path, and replaced when QuickCalc is terminated. The new working directory will be displayed on the screen.

If you don't know the current directory, enter **CHDIR "."**, or type **PRINT WORKINGDIR$**.

You can specify "**..**" or "**..**\\" or "**..***directory*" or any other valid DOS path (remember to use double-backslashes).

GUI Mode: If the path does not already exist, you have the option of creating it. You may **not** specify "**..**" or "**..**\\" . The new path replaces the path specified in the "Working Directory" box, and is the same as if you had typed it into that box, except that the **CHDIR** statement may be included in a program. . If you don't know the current directory, enter **PRINT WORKINGDIR$**. The new path is added to the list of recently used working directories. It remains in effect until you change it again.

Note: If **CHDIR** is used within a running BASIC program, the current working directory will be reset to its original path when the program terminates.

CHECKDEGREES (function)

Returns 1 if degrees is set, 0 if radians (see **DEGREES** and **RADIANS**). Allows a subroutine to function differently based on the degrees/radians setting. Allows you to save and restore that setting, if necessary.

CHECKMOUSECLICK (function)

CHECKMOUSECLICK [*no parameters*]
Returns the shape ID of the shape clicked on (if the shape was created with the **MOUSECLICK** option. (see "*Advanced Graphics*"). If nothing has been clicked since the last call to **CHECKMOUSECLICK**, it returns -1. If the user clicked somewhere other than a "clickable" shape, it returns -2. If the user typed a character, the ASCII code of that character is returned with a negative sign.

CHOOSEFONT$** (function) **CHOOSEFONT$ [*no parameters*]
Displays a font selection dialog and allows you to choose a typeface, size, bold and/or italic. **Returns the typeface [font] name**.
Shows only scalable fonts which are installed on your computer.
The point size is placed in the system variable ***choosefontsize***.
The **bold** value is placed in the system variable ***choosefontbold***, as a number from 0 to 1000 (typically 400 for normal, 700 for bold).
The *italic* value is placed in the system variable ***choosefontitalic*** (0 or 1).
You may use these values in the **SHAPE** or **TYPE** statements (see "Intermediate Graphics").
Note: If you click the **CANCEL** or [X] buttons, it will generate an error and terminate the BASIC program.

CHR$** (function) **CHR$ (*num-expr*)
Converts ASCII code to its character equivalent.
CHR$ (*x*) is valid for $0 \le x \le 255$.
CHR$ (0) returns a NULL string.

CINT (function) **CINT** (*num-expr*)
Converts a number to an integer. The fractional portion is rounded to the nearest integer (positive or negative).
(Also, see **INT** and **FIX**)

CLEAR (command) **CLEAR**
Sets all numeric variables to 0, string variables to NULL.
Undefines (undimensions) all arrays. Undefines LONGMATH variables.
Frees all string memory and LONGMATH structures.
Note: **CLEAR** is automatically performed before each **RUN**.

CLOSE (statement) **CLOSE** *file-num-1* [, *file-num-2*…]
file-num-n (numeric expressions) refer to open file(s). If a file is not open, no action is taken.
CLOSE with no parameters closes all open files.
Note: all files are automatically closed when the program terminates, unless you are **CHAIN**-ing to another program.

CLS (statement) **CLS**
(*GUI Mode Only)* Clears the console screen (window).

Does not affect the log file.

COLOR (statement) **COLOR** (*num-expr*)
(*Console mode only)*, sets FG and BG color. Use 16*(BG color) + (FG color).
0=black, 1=blue, 2=green, 3=cyan, 4=red, 5=magenta, 6=brown, 7=white, 8=gray, 9=lt blue, 10=lt green, 11=lt cyan, 12=lt red, 13=lt magenta, 14=yellow, 15=brt white. Default color = 7.

CONT (command) **CONT**
Resumes program execution while in stepping mode, usually after a **STOP** statement (or **DEBUG STEP**) or if interrupted by the **STEP** button (see "Debugging").
Note: Clicking **CONTINUE** has the same effect.
Won't work after errors. Doesn't work if a program is not paused..

CONTINUE (statement) **CONTINUE**
(new) Jumps to the end of the current **FOR** or **WHILE** loop. Does not break out of the loop, like **BREAK** does. Causes the loop to continue to execute unless its end condition has been met.

COS (function) **COS** (*num_expr*)
Cosine of an angle. The angle is assumed to be in radians, unless the **DEGREES** statement is given. (see **DEGREES** and **RADIANS**)

COSSAME (function) **COSSAME** *[no parameters]*.
(*new*) Following a **SIN** function call for a LONGMATH angle, returns the cosine of the same angle. This works very quickly, since the cosine and sine are calculated at the same time, and the cosine value is saved. If no **SIN** function call has been made, **COSSAME** will return zero.

DATA (statement) [*stmt-num*] **DATA** *value*, *value* …
May contain numbers or strings (quoted or unquoted), separated by commas. **DATA** statements must end with a colon or end-of-line.
Quoted strings may use \", \r and \n to insert quote, c/r and new-line into the string.
Unquoted strings will have blanks and tab characters stripped off front and back when they are read, and may **not** contain commas, colons, quotes, // or embedded escape sequences (\", \n or \r). They may not start with a digit, sign, or decimal point (*i.e.*, must not be confused with a number). Colon, comma, end-of-line and comment (//) will terminate the un-quoted string.
DATA statements do not need statement numbers, unless they will be referenced by **RESTORE** statements.

More than one **DATA** statement may appear on a line, but only the first one is assigned the line's statement number, for the purposes of the **RESTORE** statement.

DATA statements may appear anywhere, including inside loops, subroutines, **IF** clauses, or after the end. The statements are not executed, and are accessed *in the order they appear in the program*.

DATA statements may not be entered from the command line.

DATE$** (variable) **DATE$

Returns a string, "mm-dd-yyyy".

Date is set once when QuickCalc starts, updated with **UPDATEDATETIME**.

DAYOFWEEK$** (variable) **DAYOFWEEK$

(New). Returns Day as 3-character abbreviation, *e.g.*, "Tue". Updated along with **DATE$** and **TIME$.**

DEBUG (statement) **DEBUG** [*optional-parms*]

(new). Sets parameters for debugging. (See "Debugging").

DEBUG STEP enters stepping mode (**CONT** exits stepping mode).

DEBUG TRACE enters tracing mode.

DEBUG TRACEOFF leaves tracing mode.

DEBUG TIMER, *function* [**,OFF**] times LONGMATH functions (See "Debugging").

DEBUGPRINT (statement) **DEBUGPRINT** *variable-name*

(one variable or array item only- no expressions or constants)

(new).

For LONGMATH variables:

DEBUGPRINT displays the variable as it is stored, using six lines, maximum.

Shows the first 64 digits and at least the last 64, along with the name of the variable, number of blocks, sign, exponent, length, and normalized state. This is useful when debugging extremely long numbers, as you don't have to print the entire number.

For DOUBLE variables:

DEBUGPRINT displays the variable name and its current value with the maximum precision.

For STRING variables:

DEBUGPRINT displays the variable name, the current length and the current string value. If the string will not fit on the same line, it will start in column 1 of the next line. Trailing blanks are not visible.

You may also use **??** as a shortcut for **DEBUGPRINT**.

DEF (statement) **DEF** *function-name* ([*arg* [, *arg*]…]) = *expression* | **GOSUB** *stmt-no*

Defines and names a function that you write.

function-name is a valid variable name. This name becomes the name of the function. *name* must be unique and may not duplicate the name of any BASIC function, statement, command, constant or operator (see the **PRINTALL RESERVED** statement), and may not be the same as any variable or array name used in your program.

arg is an argument. It is a valid variable name in the function definition that is replaced by a value when the function is called. The name cannot be the same as any function, statement or command in the BASIC language. The arguments in the list represent, on a one-to-one basis, the values (parameters) that are given when the function is called. If no arguments are supplied, the parentheses are still required. Parameters given when the function is called must match the type (numeric or string) of the corresponding argument. If fewer parameters than arguments are supplied, the remaining arguments will be set to 0 or NULL string.

expression defines the returned value of the function. The type of the result of the expression must be compatible with the way it will be used or the variable to which it will be assigned. The "expression" form of the function definition is limited to one line.

stmt-no is the statement number of a subroutine (see the **GOSUB** statement). If the function is defined this way, the subroutine will be called, using the same over-riding variables (arguments). The function returns the value that the subroutine provides in the **RETURN** statement (see **RETURN** statement).

Arguments (*arg*) that appear in the function definition serve only to define the function. They do not affect program variables that have the same name. Unused arguments are not a problem. A variable used in the *expression* or subroutine does not have to appear in the list of arguments. If it does, the value of the argument is supplied (or implied) when the function is called. Otherwise, the current value of the variable is used (see "User-Defined Functions").

A function definition *expression* may include other user-defined functions, however a care must be taken to avoid a recursive definition which would result in an endless loop and cause the program to fail (see "User-Defined Functions"). The program will teminate with an error message if you attempt to nest user-defined function calls to a depth of 200.

A function definition subroutine is subject to all the rules governing **GOSUB**. Subroutines may be nested, but be careful with recursive subroutine calls to avoid an infinite loop (see paragraph above).

A **DEF** statement may appear anywhere in the program, even after the statements that call it. If the same function is defined more than once, it will cause an error.

LONGMATH numbers may be used along with DOUBLE numbers, just as with regular expressions. The type of numbers used as parameters

will affect the type of computation and the type of output. See "Working with Long Numbers" and "User-Defined Functions".

DEGREES (statement) **DEGREES**
(*new)* Causes all subsequent Trig functions to assume the angle is given or required in degrees. This remains in effect until a **RADIANS** statement is given, and is reset to radians automatically at the start of a program.

DELAY (statement) **DELAY** (*num-expr*)
Causes the program to wait for a specified number of milliseconds (between 1 and 10000). This can be used to slow a program down so that you can read the output or watch a graph being plotted. The program uses no CPU cycles while it is waiting.

DELETESHAPE (statement) **DELETESHAPE** (*shape-id*)
Deletes a shape from the graph.
shape-id is a value retrieved from **LASTSHAPEID** immediately after a shape is drawn.
See "Advanced Graphics".

DIM (statement) **DIM** *array-1* (*num_expr* [, *num-expr* …]) [,*array-2* (…)] [,…]
Dimensions arrays.
Note: Subscripts are relative to 0 *(*No *OPTION BASE).*
num_expr must be > 0.
Arrays must be dimensioned before using them.
Maximum of 5 dimensions for any array.
Arrays may **not** have the same names as scalar variables (*e.g*., *x* and *x*(2)).
All elements are initialized to zero or NULL.
Note: Use **LONG** to dimension LONGMATH arrays.

(function) *num-array* = **DIM** (*array*)
Returns the dimensions of an array in *num-array*.
num-array must be dimensioned with 6 elements, *e.g,*. **DIM** *b* (6).
The order (number of dimensions) is returned in *num-array* (0).
The size of each dimension is returned in *num-array* (1) through (5).

This is useful in a subroutine or piece of included code where the operation is dependent on the size of the array. Can be used to make sure the array doesn't overflow or make a check on subscript values.

ELSE (see **IF**).

END (statement) **END**
Ends the program. Nothing beyond the END will be executed unless it is branched to or called.

ENDTIMER (function) **ENDTIMER** *[no-parameters]*

Ends the timing of an interval and returns the count in microseconds.
The interval is the time elapsed since the last **STARTTIMER** function call.
If there was no **STARTTIMER** function call, the value returned is meaningless.

Note: Timing includes the time for interpreting the BASIC statements and all system overhead that occurs between the **STARTTIMER** and **ENDTIMER** function calls. For that reason, it is not an accurate indication of the time required to do mathmatical functions. (See **DEBUG TIMER**).

EOF (function) **EOF** (*num-expr* [, **"LINE"**]).
num_expr must be a file number, currently open for input or **RANDOM**. It is a numeric expression which will be truncated to an integer.
Returns –1 ("true") if we are at end-of-file (no data remaining), 0 if not.
The optional second parameter, **"LINE"** indicates that the next input operation will be a **LINE INPUT #**. In this case, the **EOF** function returns 0 if any records or data (including blank or zero-length records) remain in the file. **LINE INPUT #** can read such records.
If the second parameter is omitted, it is assumed that the next input operation will be an **INPUT #**. In this case, the **EOF** function skips over blanks and, for sequential files, reads subsequent records to search for more data.
For **RANDOM** files, the search stops at the end of the buffer.
(Also, see the **INPUT** function.)

ERASE (statement) **ERASE** *arrayname* [, *arrayname*] …
Eliminates arrays. Un-dimensions the array.
If the array is LONGMATH, it frees LONGMATH allocated buffers.
After doing **ERASE**, you should do **HOUSECLEAN** to free string memory and LONGMATH variable structures.

EXP (function) **EXP** (*num-expr*)
Calculates e^x.
This function can overflow:
For DOUBLE, overflows around x > 709.
For LONGMATH, overflows at around x > 2302585.

EXP10 (function) **EXP10** (*num-expr*)
Calculates 10^x.
This function can overflow,:
For DOUBLE, overflows around x > 308.
For LONGMATH, overflows at around x > 1000000.

EXPANDDATETIME (function) *datetime-array* = **EXPANDDATETIME** (*datetime-variable*).
Expands a DATETIME variable into a DATETIME array.

If no datetime-variable is given, it expands the current date/time.
Note: See "Working With Dates and Times" for definitions, examples, and descriptions of the array elements.

EXPONENT (function) **EXPONENT** (*num_expr*)

Returns the exponent of a floating-point value (DOUBLE or LONGMATH), disregarding the sign of the number.

The exponent here is the power of 10 when the number is ≥ 1 and < 10 ("scientific notation"). **Note**: this is one lower than the exponent used when a LONGMATH variable is stored.

EXPONENT (1.2) = 0. **EXPONENT** (25) = 1.
EXPONENT (.001) = -3. **EXPONENT** (-1500) = 3.
EXPONENT (**LONGPI** * 1e500) = 500.

FACTORIAL (function) **FACTORIAL** (*num-expr*)

Computes *n*! [*n*-factorial], where *n* > 0.
n will be converted to an integer (truncated) first.
If *n* is a DOUBLE, the factorial will be calculated in DOUBLE floating-point arithmetic, yielding a DOUBLE result. Maximum size for *n* is 170.
If *n* is a LONGMATH, the factorial will be calculated using long arithmetic, yielding a LONGMATH result. Maximum size for *n* is 205021, which makes *n*! close to 1 million digits long (yes, the program can do that, but it takes several minutes).
Note: Factorials get quite long. DOUBLE precision floating-point is only about 16 digits long (approx. 20!). If you want all the digits, use LONGMATH and set **INTEGER** mode (**FLOAT** is OK as long as the length remains less than *floating_pt_length*, after which you will lose digits to rounding.)

a = **FACTORIAL** (**LONG** (*n*)) will do LONGMATH calculation and then convert the result to DOUBLE (assuming *a* is DOUBLE), however, values of *n* > 170 will yield an answer too large to convert back to DOUBLE.

FIELD (statement) **FIELD** [**#**]*file-num*| *string-var-name*, *width* **AS** *string-var* [, *width* **AS** *string-var*]…

Allocates space for variables in a RANDOM file buffer or string variable.

file-num is the number under which the file was opened.
string-var-name is the name of a string variable – (not a string array or array element.)
width is a numeric expression specifying the number of character positions to be allocated to string-var.
string-var is a string variable that is used for random file access.

A **FIELD** statement defines variables used to get data out of a RANDOM buffer after a **GET** or to enter data into the buffer for a **PUT**.

A Field statement can also define variables used to extract data from a string or insert data into a string.

The statement:

FIELD #2, 10 **AS** *a*$, 30 **AS** *name*$, 40 **AS** *address*$

Defines the first 10 positions in the buffer as the string variable *a*$, the next 30 as *name*$, and the next 40 as *address*$.

The statement:

FIELD *abc$*, 10 **AS** *a*$, 30 **AS** *name*$, 40 **AS** *address*$

Defines the first 10 positions in the string variable *abc$* as the string variable *a*$, the next 30 as *name*$, and the next 40 as *address*$.

FIELD does not actually place any data into the buffer or define the string. This is done by assigning values to those variables, reading data into them, or using **LSET** and **RSET** statements. **FIELD** does not "remove" data from the file/string, either. Information read from the file/string with the **GET** statement is read from buffer/string by simply referring to the variables defined in the **FIELD** statement (see "Working with Files").

A **FIELD** statement may define up to 10 different variables. You may re-define the buffer/string by specifying additional **FIELD** statement with the same *file-num* or *string_var_name* and different variables and positions. This has the effect of having multiple field definitions for the same data. You can skip over a block of characters by using *width* **AS SPACE$**, which doesn't count toward the total number of variables.

You may specify a total of 40 **FIELD** statements in a program. They may appear anywhere in the program and in any order. **FIELD** statements may not be entered from the command line.

FILESIZE (function) **FILESIZE** (*file-num*)
Returns the file size, in bytes, of an **open** file.
file-num is a numeric expression which must refer to the file number of a currently open file.
If the file is open for OUTPUT or SERIAL, this function returns zero.

FIX (function) **FIX** (*num-expr*)
Truncates a number to an integer (| *x* | is moved "down", closer to zero)
Truncates digits right of the decimal point - does not round!

(Also, see **CINT** and **INT**.)

FLOAT (statement) **FLOAT** [*num-expr*]

Sets LONGMATH calculation to floating-point mode.
num-expr specifies the floating point length (range 20 to 1000000).
If *num-expr* is not provided, the last value set is used. Default is 50.

FOR / NEXT (statements) **FOR** *num-var* = *num-expr* **TO** *num-expr* [**STEP** *num-expr*] ….. **NEXT**.

Loops may be nested. *Each loop requires a separate* **NEXT**.
NEXT *may not contain a variable name*.
Loops are parsed for structure. The **NEXT** must be physically the last statement in the loop. An inner loop must end before the outer loop.
Branching into or out of a loop is not allowed. Branching around a loop is OK. Statement numbers are checked before the program runs.
Limit (**TO** value) and increment (**STEP** value) are calculated once before the loop begins.
The entire **FOR** … **NEXT** loop may be on one line (if desired). If the statement following the **TO** or **STEP** begins with a variable or statement which could also be a function (like **INPUT**), precede it with a colon (*e.g.*, **FOR** i =1 **TO** 10 **:** $j = j + i$ **NEXT**) to avoid confusing the it with part of the expression for **TO**.

FORMATDATETIME$** (function) **FORMATDATETIME$ (DATETIME *variable*)

Generates a formatted string for a date and time, for example:

```
Wed  Apr  08  12:35:36.123445  DST          1942
```

FORMATDATETIME$ ( ). Generates a formatted string for the current date and time.

GET (statement) **GET #***file-num* [, *record-number*]

Reads data into the random-access file buffer from the file at *record-number*.
record-number is relative to 1. If *record-number* is omitted, the next record number following the last **GET** is used.
If you read to a record number which is beyond the end of the file, an error will occur. The contents of records that were never **PUT** will be returned as blanks.
Following the **GET**, the data from the desired record is placed in the buffer so that it can be referenced by **INPUT #, LINE INPUT #** or by using **FIELD** variables. The pointer is reset so that **INPUT #** will begin reading at the beginning of the buffer.
If you want to reset the pointer and re-**INPUT** the record, simply re-issue the **GET** to the same record number.

GOSUB (statement) **GOSUB** *stmt-num*

```
....
stmt-num   (beginning of subroutine)
....
RETURN
```

GOSUB transfers control to the statement at *stmt-num*.
The subroutine runs until a **RETURN** statement is encountered, then control resumes at the next statement following the **GOSUB**.
Subroutines may be nested, but it is not safe to overlap them.
No statement-number checking is there to prevent branching out of the subroutine (due to the requirement for multiple **RETURN** statements), so be careful.

GOTO (statement)

GOTO *stmt-num*

Branches unconditionally out of the normal program sequence to a specified line number.
stmt-num must refer to the line number of a statement in the program. You may not branch into or out of a **WHILE-WEND** loop or a **FOR-NEXT** loop. Statement numbers are checked to prevent this.

GRAPH (statement)

GRAPH [parameter [, parameter [, …]]]

GRAPH with no parameters or the first **GRAPH** statement starts a new graph. Parameters are:

AXES	Draws x-y axes and numbers.
AXISWEIGHT=*num-expr*	line thickness of axes and borders, in points.
BGCOLOR=[(*r,g,b*) | *num-expr*]	Sets the background color.
BORDER	Draws a border around your graph.
CLEARANCE=*num-expr*	When auto-scaling, leaves a space of *num-expr* inches between the extremes of the plotted points and the border. *num-expr* may be 0 to .25. Default is 0.1.
END	Ends the graph and closes the graph window
EQUALSCALES	force H and V scales to be the same
GRID	Draws grid lines at the tick points.
GRIDCOLOR=[(*r,g,b*) | *num-expr*]	Sets the grid color.
LOCKSCREEN=[**0**, 1]	prevents unintentional dragging/zooming.
HEIGHT-*num-expr*	height in inches
HMARGIN=*num-expr*	horizontal margin, in inches.
HOFFSET=*num-expr*	horizontal offset, in data units.
HSCALE=*num-expr*	horizontal scale, in data units/in.
HSCALETYPE=[LOG | LINEAR | DATETIME]	log, linear or DATETIME scale on *x*-axis.
INCLUDEXAXIS=[0 or 1]	Autoscale includes [1] or does not include [0] the x-axis on the graph.
INCLUDEYAXIS=[0 or 1]	Autoscale includes [1] or does not include [0] the y-axis on the graph.

JPEG	Generates a JPEG file of the graph.
LANDSCAPE	sets 10 in. wide, 7.5 in. high
LISTPARMS	Lists all the **GRAPH** parameters currently in effect.
MAXIMIZE	Maximizes the graph window (same as the "maximize" button). Allows you to start the graph maximized. You should set the size of the graph before maximizing it.
METAFILE	Generates a metafile (.emf) from the graph.
NOAXES	Suppresses drawing x-y axes & numbers.
NOBORDER	Does not draw a border.
NOGRID	Suppresses drawing the grid.
NUMBERXAXIS=[0 or 1]	Does [1] or does not [0 display ticks and numbers along the *x*-axis.
NUMBERYAXIS=[0 or 1]	Does [1] or does not [0 display ticks and numbers along the *y*-axis.
PORTRAIT	(default) sets 7.5 in. wide, 10 in. high.
PRINT	Prints the graph.
VMARGIN=*num-expr*	vertical margin, in inches.
VOFFSET=*num-expr*	vertical offset, in data units.
VSCALE=*num-expr*	vertical scale, in data units/in.
VSCALETYPE=[LOG | LINEAR | DATETIME]	log, linear or DATETIME scale on *y*-axis.
WIDTH=*num-expr*	width in inches

(See the document "Intermediate Graphics" for more information.)

Note: **GRAPH** is not valid in CONSOLE mode.

HALF (function) **HALF** (*num-expr*)

Computes half of a number.
This function is about ten times faster than dividing by 2, and is useful when working with extremely long LONGMATH numbers. It is also compatible with DOUBLE numbers, but without the time savings.

HELP (statement) **HELP** [*topic-number*]

Designed primarily for the Console interface, **HELP** with no topic number lists the available Help topics (documents).
Typing **HELP** followed by the number of a topic from the list causes that Help document to be opened in Adobe Reader.
GUI-mode users will find the HELP button more useful, and it accesses the same documents.

HEX$** (function) **HEX$ (*num-expr* [, *fill*])

Converts a number to its hexadecimal equivalent, and places it in a string.
Range is that of a long integer (-2147483648 < x < 4294967295).

New: There is an optional numeric second parameter *fill*.

fill= -1: (no padding-default). Length is variable: "B20"

fill=0: left-fill the number with zeros. Length is 8 chars: "00000B20"

fill=1: (blank padding). Length is 8: " B20"

fill=2: (floating point). Displays the internal floating-point representation of the number. Length is 16.

Note: This function is for DOUBLE values. If you want to look at the internal representation of LONGMATH numbers, use the **DEBUGPRINT** statement.

For *fill* = -1, 0 and 1, the number is first converted to an integer, as in **FIX** ( ), and then into hex. For Fill=2, the internal floating-point representation (DOUBLE) is used. The argument may be a constant or an expression. If the argument is a LONGMATH number, it will be converted into a DOUBLE, if possible.

Note: The string produced by **HEX$** (*number*, 2) can be converted back into a DOUBLE by preceding the string with "0x" and assigning it to a DOUBLE variable or using it in a numeric expression, *e.g.*

a = "0x" + **HEX$** (*b*, 2).

HEXCONVERT (function) **HEXCONVERT** (*num-expr*)

Converts the *num-expr* to LONGMATH, and then into a long hex string (up to floating_pt_length digits to the right of the decimal point). If the resulting length is <= 255, the result may be assigned to a string variable or used anywhere a string is allowed. If it is longer, it will be truncated (in the fractional part) to 255, or it may be assigned to a string array, similar to the way it is done with LONGMATH numbers.

HEXCONVERT (*string-expr* | *string-array)*

Converts the *string-expr* or *string array* into a LONGMATH number, which can be used in a numeric expression. Precision is determined by floating_pt_length. It can be assigned to a DOUBLE variable if it is within the range of a DOUBLE.

Note: **HEXCONVERT** does not require integers. Data to the right of the decimal point is also converted (up to the floating_pt_length).

See "Number Representation and Conversion" for more information on this function.

HOUSECLEAN (command) **HOUSECLEAN**

Re-organizes the string storage and frees unused string space. Also frees **LONGMATH** variables released by **ERASE**.

Housekeeping is (will be) done automatically if the string storage fills up.
Note: A large block of storage (equal to the string storage space) must be available in order to re-organize the string storage. This function will fail if the memory cannot be allocated.

IF (statement) **IF** *expression* **THEN** *clause* [**ELSE** *clause*]
IF *expression* **GOTO** *stmt-num* [**ELSE** *clause*]
IF *expression* **ELSE** *clause*
Entire **IF** statement must be on one line.
Complete **WHILE** and **FOR** loops may be part of an **IF** clause (if you can fit them all onto one line).
No commas between items.
IF statements may be nested. The inner statement may need an **ELSE** to avoid confusion with the outer statement. If there is no outer **ELSE** clause, follow the inner **ELSE** with a colon.
Note: **IF** *expression stmt-num* is not allowed.

INCLUDE (statement) **INCLUDE** *file-spec*
Includes the program specified by *file-spec* in the current program **following the current statement line**. This is done before the program is run.
file-spec (the file specification) must be a string. If the *file-spec* contains a colon, backslash(es) or embedded blanks, it must be in quotes – (see "Strings"). You may use any file specification that works. If the path is in quotes, you must use a double backslash (\\) wherever a backslash is desired. (Don't use **..**\). If a complete path is not specified, the *file-spec* is appended to the **path from which the original program was loaded**.

This function is useful if you have a collection of definitions, **DATA** statements or array dimensions that are used in many different programs. Simply group the statements together as a program and then **INCLUDE** them.
Note: The result of the inclusion must be syntactically correct. No duplicates (statement numbers, functions or array definitions). The updated program must fit in the space allocated to load programs as set in the "Customize" dialog.
Note: The **INCLUDE** statement may appear anywhere in the program. It does not get executed at run time. If the **INCLUDE** statement is at the end of the program, it is the same as an **APPEND** statement.
Note: An included program may include or append another program, if desired (see **APPEND**). Be careful to avoid a recursive loop of **INCLUDE**s (program loads itself).

**INKEY$** (function) **INKEY**$ [no parameters]

Waits for the user to type a keystroke and returns that character as a 1-character long string. The character must be a printable character (not a function key or scroll key.) A carriage return [ENTER] is not required. The BASIC program is suspended until the user types a character, although the graph window may still be controlled with the mouse.

This function differs slightly from the IBM BASIC version, since Windows ignores all keystrokes before the function is called, therefore you can't test to see if a key has been pressed; you just have to wait for it.

Note: If you want to test if a key is pressed when a graph is active, you can use the **CHECKMOUSECLICK** function (see "Advanced Graphics")

The **TAB** key and the four arrow keys have special functions when a graph is active and the graph window has the focus. These keys are not passed back to **INKEY$.** In the main QuickCalc window, the keystrokes are also ignored when doing **INKEY$.**

You can break out of an **INKEY$** loop by clicking **STEP** and then typing the character that the program is waiting for.

INPUT (statement) **INPUT** [;]["*prompt*";] *variable* [,*variable* …]

The first optional semicolon indicates no c/r will be printed after the data is entered (GUI mode only).

"*prompt*" is a constant string.

Second semicolon may be replaced with a comma. In Console mode, the comma means don't follow the prompt with a question mark.

In GUI mode, data is entered in the command window, which is labeled "Data:" and turns pink when data is requested.

Data values must be separated by commas, and must agree in number and type with the variable list. Constants only – no expressions or variables.

Number strings may be assigned to string variables. Number strings will be converted, if possible, to match the destination variable.

See "Introduction" and **DATA** (above) for format of string data values.

This version: values are assigned to variables as they are read. If an error occurs, "**?Redo from Start**" is printed, variables will be over-written with the new values.

Note: Also, see **INPUTDIALOG**.

INPUT (function) **INPUT** (*file-num*).

Tests for end-of-file before doing an **INPUT #** in order to avoid "End-of-File" errors.

Tests if a message is ready to be read (for SERIAL files). It returns 1 if a message is ready, 0 if not.

File-num represents a file which is open for INPUT, RANDOM or SERIAL. It is a numeric expression which will be truncated to an integer.

End-of-file is determined when no more data is left, after skipping over blanks, c/r, or l-f, OR when the end of a random-access file buffer is reached.

This function differs from **EOF** (*file-num*) in that it returns 0 for "no more data" (eof), and if we are not out of data, it returns the next data item type:

1 = string
2 = number
3 = invalid type

Type 3 may only be read using **LINE INPUT**.

Note: If the next operation you plan to do is a **LINE INPUT #**, you should use **EOF** (*file-num*, **"LINE"**) instead of **INPUT** (*file-num*), since the **INPUT** function will skip over blanks and zero-length records which you may want to read.

INPUT

(statement)

INPUT # *file-num, list-of-variables*

file_num represents a file which is open for INPUT or SERIAL It may be followed by a comma or semicolon.

list-of-variables is the same as for **INPUT** *(See* **INPUT***).* If a mismatch or EOF error occurs, the program will be terminated.

If *file-num* refers to a RANDOM file, data is read from the buffer filled by the last **GET** from that file. **INPUT #** may not attempt to read beyond the end of the buffer.

If *file-num* refers to a SERIAL file, extra variables or extra data items will be ignored (see "Working with Files").

Note: you can use the **EOF** ( ) or **INPUT** ( ) functions to test for end-of-file before reading.

Note: you can use the **INPUT** ( ) function to see if a SERIAL message is ready before reading, to avoid time-out errors.

INPUT %

(statement)

INPUT % *string_expr; list-of-variables*

string_expr is a string containing data that you wish to read from.

list-of-variables is the same as for **INPUT** *(See* **INPUT***).* If a mismatch occurs, the program will be terminated.

This function takes a string and reads data from it as if it were a file buffer.

There is no error condition for too many or too few data items for the number of variables in the list. Unused variables will not be changed (you should preset variables to default values.) Extra data items will be ignored. Subsequent **INPUT %** statements will start again at the beginning of the string.

You can use this function to extract data from a string you generated, read, or calculated. You can also use it to parse a string read by **LINE INPUT** in more than one way.

INPUTDIALOG (statement) **INPUTDIALOG** [*title* **;**] *variable-name-1*, *variable-name-2*, …

Displays a dialog box allowing you to input the desired variables.
title (optional) The title or caption for the dialog box. Default is "Enter Data". must be a string constant (in quotes) or variable. If used, it must be followed by a semicolon.
variable-name-1, variable-name-2, … is the list of variables to input (maximum of six).
Note: There is no return code. If the user clicks "OK" or ENTER, the values are saved. If you click the [X] or press ESC, the program is terminated.
See the document "Advanced Features of QuickCalc" for more information.
Also, see **UPDATEDIALOG**.

INPUTDIALOG (function) rc = **INPUTDIALOG** ([*title*,] *string-array*)

title (same as for the statement form, above). Follow it with a comma.

string-array Specifies the name of the string array, (usually dimensioned (*n*, 5), which a one- or two-dimensional array containing the data to define the dialog box and the variables you want to enter.

Format of each row of the string array:
variable-name, *prompt*, *row*, *start-column*, *width*

variable-name is the name of the variable to receive the data, or a number (1-99) to define a button.
prompt is the text to display above each edit field or on the button.
row is the row (0-5) in the dialog where this field appears.
start-column is a percentage (0-95) of the dialog width.
width is the width of the edit field or button, as a percentage (5 to 100) of the dialog width.

Return code is 0 of user clicked "OK" or pressed ENTER, -1 if user "cancelled" (clicked the [X] or pressed ESC), or the number of the user-defined button.
See the document "Advanced Features of QuickCalc" for more information.
Also, see **UPDATEDIALOG**.

INSTR (function) **INSTR** ([*n*],*x*$, *y*$)

Searches for the first occurrence of string *y*$ in *x*$ and returns the position at which the match is found. The optional offset *n* sets the position for starting the search in *x*$.
n is a numeric expression from 1 to 255.
If *y*$ is not found in *x*$, **INSTR** returns 0.
If *n* > **LEN** (*x*$) or *x*$ is NULL, **INSTR** returns 0.
If *y*$ is NULL, **INSTR** returns *n* (or 1 if *n* is not specified.)

INT (function) **INT** (*num-expr*)
Largest integer less than or equal to *x*. ("floor" function).
Positive numbers are truncated at the decimal point.
Negative numbers which are not integers are moved "down" (more negative) to the next integer.
(Also, see **CINT** and **FIX**.)

INTEGER (statement) **INTEGER**
Sets LONGMATH calculations to Integer mode.
Does not change the format of any existing LONGMATH numbers.
See "Working with Long Numbers" for the differences between Integer and Floating Point mode.

LEFT$** (function) **LEFT$ (*x$, n*)
Returns the leftmost *n* characters of *x$*.
x$ is a string expression.
n is a numeric expression from 0 to 255, and specifies the maximum number of characters in the result.

LEN (function) **LEN** (*string-expr* | *string array*)
Returns the number of characters in the string or string array.

LINE INPUT (statement) **LINE INPUT** [;] ["*prompt*";] *string-var.*

Reads an entire line (up to 255 characters) from the keyboard into a string variable. All characters are read, including trailing blanks and delimiters.
The first semicolon (optional) means don't start new line after typing input (GUI mode only).
Trailing blanks are removed.
Input is terminated when [enter] is pressed. Position of cursor when [enter] is pressed is unimportant – all characters on the line are entered.

LINE INPUT # (statement) **LINE INPUT #***file-num*, *string-var*.
Reads an entire line from the file or **RANDOM** file buffer (maximum length is 255 characters) into a string variable. The line is not interpreted or broken into fields, but is read exactly as it was in the file.

Carriage-returns and line-feeds are removed from the end of the record.

For SERIAL files, it reads the entire message from the SERIAL file buffer. Escape sequences are not converted.

If a line has been partially read by a previous **INPUT #** statement, the remainder of the line will be read in this statement.

If the remainder of the line (or **RANDOM** file buffer) is longer than 255 characters, the next 255 are taken and the rest of the line is available to be read by subsequent **LINE INPUT #** or **INPUT #** statements.

Note: you can use the **EOF** ( ) function to test for end-of-file before reading and avoid "end-of-file" errors.

Note: For SERIAL files, you can use the **INPUT ( )** function to test is a message is ready to read, to avoid time-out errors.

LIST (command) **LIST**

Lists entire program on screen. (Not for editing).

For debugging, also lists hex offset within program and statement number and **DATA** statement tables.

Note: This function is not needed simply to list or print your program. You can do that from within Notepad.

LOAD (command) **LOAD** *file-spec*

Loads program and checks for syntax. Does not run it (use **RUN**).

file-spec (the file specification) must be a string. If the *file-spec* contains a colon, backslash(es) or embedded blanks, it must be in quotes – (see "Strings"). You may use any file specification that works. If the path is in quotes, you must use a double backslash (\\) wherever a backslash is desired. (Don't use **..**\). If a complete path is not specified, the *file-spec* is appended to the **current working directory**.

You can change the current working directory from within the program with the **CHDIR** statement.

This command is only needed if you wish to load and check a program without running it. In GUI mode, it is faster to use the RUN button.

LOG (function) **LOG** (*num-expr*)

Natural logarithm.

num-expr must be greater than zero.

LOG10 (function) **LOG10** (*num-expr*)

(*new)* Common (base 10) logarithm. .

num-expr must be greater than zero.

LONG (function) **LONG** (*num-expr* | *string-expr* | *string-array*)

Converts just about anything into a LONGMATH value.

Accepts any numeric expression and assures that the result is LONGMATH. This is useful for functions like **SQR** with behave differently depending on whether their argument is DOUBLE or LONGMATH.

Also accepts string expressions and converts them, if possible, similar to **VAL**, but works on long strings and numbers. For example, you could say *y* = **LONG** ("1.5e" + **STR$** (1000)).

LONG will also accept and convert the **string array** form of a long number (see "Number Representation, Assignment, and Conversion").

(statement) **LONG** (*var-name* | *array-name* (*num_expr* [, *num-expr* …]) [, ….]

Defines a variable to be LONGMATH, or dimensions a LONGMATH array.

Variables and arrays may be intermixed on the line.

Names must **not** end in $.

Variables must not have been used or referenced previously, as that implicitly defines them as DOUBLE.

Arrays must not have been previously dimensioned (see **ERASE** and **CLEAR**).

LONG "ALL" [**OFF**]

Sets [or clears] a mode where the program uses only LONGMATH variables and does all operations and functions using long arithmetic. (See "Using Long Numbers"). In this mode, variables do not need to be declared as LONG.

LONGE (function) **LONGE** [no parameters]

Calculates and returns (LONGMATH) e, the base of the natural logarithms.

This value is calculated at the current floating-point length when it is referenced, and saved for subsequent use. If a longer value is requested, it will be re-calculated and saved. If a shorter or same-length value is requested, the saved value will be used (shortened and rounded if necessary).

LONGPI (function) **LONGPI** [no parameters]

Calculates and returns (LONGMATH) π.

This value is calculated at the current floating-point length when it is referenced, and saved for subsequent use. If a longer value is requested, it will be re-calculated and saved. If a shorter or same-length value is requested, the saved value will be used (shortened and rounded if necessary).

LSET (statement) **LSET** *string-var* = *string-expression*

string-var is the name of a variable, normally one defined in a **FIELD** statement. (See **FIELD** statement). It may not be a string array, but may be a string array element.

string-expression will be placed in the field (or "string variable") identified by *string-var*, typically in preparation for a **PUT** statement.

If *string-expression* is shorter than the width specified for *string-var* in the **FIELD** statement, **LSET** left-justifies the string in the field (spaces are used to pad the extra positions). If *string-expression* is longer than *string-var*, characters are dropped from the right.

Numeric (DOUBLE) values **must be** converted to strings before they are **LSET** (see **STR$** function). LONGMATH numbers shorter than 255 characters may be assigned to strings directly.

See "Working with Files" for more information on using RANDOM files.

While it is intended for use with field variables, **LSET** can also be used for ordinary string variables. **LSET** differs from an assignment statement in that the current length for the destination variable is preserved, and the string expression is truncated or padded to fit in that length. If the destination string variable has not been used yet, it will have a zero length and **LSET** will not place anything in it.

MAKEDATETIME (function)

datetime-variable = **MAKEDATETIME** (*month, day, year, hour, minutes, seconds, dst*). Converts the date and time specified into a DATETIME variable.
datetime-variable= **MAKEDATETIME** ( ). Converts current date and time into a DATETIME variable.
datetime-variable = **MAKEDATETIME** (*datetime-array*). Compresses the DATETIME array into a DATETIME variable.

Note: See "*Working With Dates and Times"* for more information.

MESSAGEBOX (function) **MESSAGEBOX** (*message-string*, *button-1-string* [, *button-2-string* [, *button-3-string*]]

All parameters are string expressions. If you want to display a number, put it in quotes or use the **STR$** function to convert it.

This function displays a Windows-style Message Box on the screen, allowing you to make a selection by clicking one of the buttons. You have control over the text displayed and the caption on each button. The Message Box will have 1, 2 or 3 buttons, depending on the number of parameters provided. You must have a message string and at least one button string.

The function returns the number of the button you clicked (1 is the leftmost button). If you click the "**X**" button in the upper right corner, the function will return -1.

The BASIC program is suspended until you reply to the Message Box. If you click the "**Step**" button before leaving the Message Box, the program will enter stepping mode as soon as you click a Message box button. This is useful if you find yourself in a program loop that keeps displaying the Message Box. Of course, you can also click "**Kill Program**" which will destroy the Message Box immediately and terminate the BASIC program.

Note: **MESSAGEBOX** has the same effect as printing a message to the screen and inputting a number as a reply. It just looks better, and is easier to code.

Note: You could precede the **MESSAGEBOX** function with a **BEEP** to alert you that a message is requesting a response.

Note: You should keep the strings for the buttons **short** (typically less than 15 characters). Strings do not wrap around on the buttons, and text could be lost. It is best to use longer strings to ask the question and short strings like "yes" and "no" for the buttons.

The message string will wrap, and if you want to force a new line, you may insert the new-line character (\ n) into the string.

MID$

(function) **MID$** (*x$, n* [*, m*])
(statement) **MID$** (*v$, n* [*, m*]) = *y$*

Function: returns the requested part of a given string.
Statement: replaces a portion of one string with another string.

x$ is a string expression.
n is a numeric expression from 1 to 255.
m is a numeric expression from 0-255.
v$ is a string variable or array element.

Returns a string of length *m* characters from *x$* beginning with the *n*th character. If *m* is omitted, or if fewer than *m* characters are to the right of the *n*th character, all rightmost characters beginning with the *n*th character are returned. If *m* = 0 or n > **LEN**(*x*), **MID**$ returns a NULL string.

For the statement form, characters in *v$*, beginning at position *n*, are replaced by the characters in *y$*. The optional *m* refers to the characters from *y$* used in the replacement. If *m* is omitted, all the characters in *y$* are used. If *m* > **LEN** (*y$*), only **LEN** (*y$*) characters will be changed. The length of *v$* does not change.

MOD (function) **MOD** (*num-expr*, *num-expr*)

(New). **MOD** (*x*, *y*) is the same as *x* **%** *y*.

It calculates the floating-point remainder *f* of *x* / *y* such that *x* = *i* * *y* + *f*, where *i* is an integer, *f* has the same sign as *x*, and the absolute value of *f* is less than the absolute value of *y*.

It works with DOUBLE and LONGMATH numbers.

y may not be zero.

NEXT (see **FOR**).

ON-GOSUB (statement) **ON** *num-expr* **GOSUB** *stmt-no* [*, stmt-no*] **…**

Calls the subroutine at one of several specified line numbers, depending on the value of an expression. (see **GOSUB**)

num-expr must be from 0 to 255. It will be truncated to an integer.

If *num-expr* is 3, for example, the third statement number will be used.

If *num-expr* is 0 or greater than the number of statement numbers in the list, the **GOSUB** will not be executed.

ON-GOTO (statement) **ON** *num-expr* **GOTO** *stmt-no* [*, stmt-no*] **…**

Transfers control (branches) to one of several specified line numbers, depending on the value of an expression. (see **GOTO**)

num-expr must be from 0 to 255. It will be truncated to an integer.

If *num-expr* is 3, for example, the third statement number will be used.

If *num-expr* is 0 or greater than the number of statement numbers in the list, the **GOTO** will not be executed.

OPEN (statement) **OPEN** *mode*, *file-number, file-spec* [,*recl* [,*baud* [,*parity* [,*stop*]]]]

mode = constant: OUTPUT, INPUT, APPEND, RANDOM, or SERIAL *(no default).*
Does not need to be in quotes (for statement form).

file_number = number expression between 1 and 5. *Do not use #.*

file_spec = string expression: [path and] name of file to be opened. If not absolute, it is relative to the **Current Working Directory**, regardless of where the BASIC program was loaded from.
For SERIAL files, this is the name of the COMM port, *e.g.*, "COM1".

recl = number expression:
record length [**required**] for random files.
cutoff length for sequential output file records, max record length for sequential input file records. Default is 514. The length includes the terminating c/r and l-f symbols.

For SERIAL files, it is the buffer size, or the longest record to send or receive.

baud= number expression: for SERIAL files, the baud rate for transmitting/receiving. Default is 9600.

parity= string expression: for SERIAL files, the desired type of parity checking. Valid settings are: "**NO**", "**ODD**", "**EVEN**", "**MARK**" or "**SPACE**". Default is "**NO**".

stop = number expression: FOR SERIAL files, the number of stop bits used following each character. Valid values are 1, 1.5, and 2. Default is 1.

(function) Same parameters as **OPEN** statement.
mode must be a **quoted** string constant.
Returns 0 if successful, <0 if error occurs.
Error Codes:
-1 = Invalid mode for Open
-2 = Invalid file number for OPEN
-3 = File number is in use
-4 = Could not allocate buffer for file
-5 = OPEN file failed
Does not print error messages for file open.
Does print error messages if parameters are incorrect.
Note: Do not skip parameters by using successive commas. This will cause an error. You can leave off parameters from the end of the list and accept the defaults.

PLOT

(statement) **PLOT** [*parameter* [, *parameter* [, …]]]
A **PLOT** statement that doesn't specify a point, or includes a parameter other then **X**= and **Y**= will begin a new plot.
Optional parameters are:

LINEWEIGHT=*num-expr*	line thickness, in points
COLOR=[(*r, g, b*) | *num-expr*]	(*num-expr's)r* = red color (0-255), *g* = green color, *b* = blue color
SORTEDX	sort points by x-value (default)
SORTEDY	sort points by y-value
UNSORTED	do not sort points
AVERAGE=*num-expr*	moves points closer (0-10) to their average
SMOOTH=*num-expr*	smoothes (0-10) the plot using Bezier curves.
X=*num-expr*	x-coordinate of point to plot
Y=*num-expr*	y-coordinate of point to plot

(See the document "Intermediate Graphics" for more information)

PRINT (statement) **PRINT** [*list-of-expressions*] [;]

Prints to the screen.

Items (expressions) to be printed must be separated by comma or semicolon (not a space).

Print zones default to 14 chars (the value in the constant DEFAULTPRINTTABS). You can override this by setting a value in a variable PRINTTABS. Set PRINTTABS to 0 to revert back to the default.

Numbers will always be printed with a trailing space.

Numbers are formatted using the string DEFAULTPRINTFORMAT$$, which has the value "%13.6g" (c-type).

You can override this by setting a string into a variable PRINTFORMAT$$. It may be a c-type, basic-type (*e.g.*, "####.##"), the string "max" or "ENG*xxx*". This same format specification is also used in the **STR$** function. You can also override it using the **PRINT USING** statement. See "Number Formatting", in the Introduction.

TAB is implemented, but *SPC is not* (use **SPACE$** instead)

Number strings are printed the same as character strings (left-justified).

If the list of expressions ends in a semicolon, the next print statement will begin where this one left off (no new line, unless the line was full).

You can also use **?** as a shortcut for PRINT.

PRINT # (statement) **PRINT #***file-num*; [*list of expressions*][;]

Works the same as **PRINT**, except prints to a **file**, random-access file buffer or SERIAL port.

file-num is a numeric expression.

file-num must refer to an open file, opened for OUTPUT, APPEND, RANDOM or SERIAL.

file-num must be followed by ; *(or comma)* if items follow

If *file-num* refers to a **RANDOM** file, data is written to the buffer in preparation for a **PUT** to that file. **PRINT #** may not attempt to write beyond the end of the buffer. If the *list of expressions* does not end in a semicolon, the remainder of the buffer is padded with blanks. If you do a second **PRINT #** or **WRITE #** to the buffer, it will over-write the first, unless the first list-of-expressions ended in a semicolon. If you want to clear the buffer, just execute a **PRINT #** statement with no parameters.

If *file-num* refers to a SERIAL file, a complete message is written to the serial port. If the statement ends with a semi-colon, you must issue another **PRINT #** or **WRITE #** to complete the line. Writing beyond the buffer length will cause an error.

PRINT % (statement) **PRINT %** *string-var*; [*list of expressions*][;]

Similar to the **PRINT** statement, except that the output goes to a string variable.

Some differences are:

If a line terminates with a semicolon, data will be placed in the destination string, but will also be retained in the buffer for subsequent **PRINT%** or **WRITE%**.

The maximum line length is 255 characters. If a data item will not fit on the remainder of the line, it will not be copied into the string, nor will any subsequent items in the list. If the first item is a LONGMATH variable, and is longer than 255, only the first 255 characters will be printed. The data will not wrap around or continue to the next line – (there is no next line). You should stay within the 255-character limit.

If a **TAB** ( ) references a column previous to the current print position, it will *not* skip to the next line (there is no next line). The string will be terminated at that point.

PRINT % is useful for formatting strings, particularly those that are going to be used later in reports or graph **TEXT** statements.

Note: **PRINT%** and **WRITE%** statements may be intermixed to create a string. Use the semicolon at the end of the list to show that the list is continued in the next statement.

Note: **PRINT%** statements may also use the **USING** parameter, e.g., **PRINT%** *abc*$; **USING** "*format-spec*"; *list-of-items*. Different **PRINT%** statements may use different format specifications.

Note: There are other ways to format strings, including string concatenation (**+**), **MID$**, **LSET** and **RSET** with **FIELD** statements, **STR$**, **STRING$**, **SPACE$**, **FORMATDATETIME$**, **HEX$**, **HEXCONVERT**, **DATE$**, **TIME$**, etc. You may use all these tools to get the formatting you like, or stick with the functions with which you are most comfortable.

PRINT USING (statement) **PRINT USING** *v*$; [*list of expressions*][;]

Works exactly like the **PRINT** statement except it uses an over-riding format string for numbers. See "Number Formatting", in the Introduction.

v$ is a format string (c-type or basic-type), "ENGxxx" or "max", which is used to format numbers instead of the default or override format strings. There is a special format string for LONGMATH numbers (see "Working With Long Numbers").

v$ must be followed by a semicolon.

No formatting of string items.

PRINT # USING (statement) **PRINT #***file-num*; **USING** *v$*; [*list of expressions*][;]

Works the same as in **PRINT** and **PRINT USING**.

PRINT % USING (statement) **PRINT %** *string-var*; **USING** *v*$; [*list of expressions*][;]

Works the same as in **PRINT** % and **PRINT USING**.

PRINTALL (statement) **PRINTALL** [*what*]

(New). Lists the current values of all data items.
Only non-zero numeric values or non-null strings are printed.
what is an optional parameter:

VARIABLES	prints only simple variables,
ARRAYS	prints only array items,
MEMORY	prints a summary of memory usage.
RESERVED	lists all the reserved words (functions, statements, etc.), which cannot be used as variable names.

Note: **PRINTALL** can generate a lot of output, so it is best to use it with the log file opened.

PROGRAMDIR$** (function) **PROGRAMDIR$ *[no parameters]*

Returns a string containing the directory from which the BASIC program was loaded. This is the directory which will be used as the default for **APPEND**, **INCLUDE** and **CHAIN** statements.
This string may be saved and used later in a **CHDIR** statement, if desired, or used to construct a path for opening files.
Note: If this function is used from the command line when a BASIC program is not running, it will return the directory from which QUICKCALC was loaded.

PUT (statement) **PUT #***file-num* [*, record-number*]

Writes data in the random-access file buffer to the file at *record-number*.
record-number is relative to 1. If *record-number* is omitted, the next record number following the last **PUT** is used.
If you write to a record number which is beyond the end of the file, the file size is increased up to and including the specified record. The contents of the intervening records is undefined.
Following the **PUT**, the data remains in the buffer so that it can be **PUT** to another record number, if desired. The pointer is reset so that **PRINT #** and **WRITE #** will begin writing at the beginning of the buffer.
If you do a second **PRINT #** or **WRITE #** to the buffer, it will over-write the first, unless the first list-of-exoressions ended in a semicolon. If you want to clear the buffer, just execute a **PRINT #** statement with no parameters.

RADIANS (statement) **RADIANS**

New. Causes all subsequent Trig functions to assume the angle is given or required in radians. This is the default. This remains in effect until a **DEGREES** statement is given.

RANDOMIZE (statement) **RANDOMIZE** *num-expr*
RANDOMIZE TIMER
Seeds the random-number generator.
num_expr may be any positive number >= 1 and <= 4294967295.
Only the integer portion of *num_expr* is used.
The resulting pseudo-random sequence generated by calls to the **RND** function will generate the same sequence if the same value is used for **RANDOMIZE**.
If **RANDOMIZE** is not called, it is the same as if you coded **RANDOMIZE** 1.
If you use **RANDOMIZE TIMER**, a different value, based on the sysyem clock, will be used each time you run the program.

READ (statement) **READ** *variable [, variable…]*
Reads values from **DATA** statements and assigns them to variables. (See **DATA**, **RESTORE** and **INPUT)**.
Numbers may be read into DOUBLE or LONGMATH variables, provided they are within the proper range.
Just about anything, including numbers, can be read into string variables, as follows:
Quoted strings may use \", \r and \n to insert quote, c/r and new-line into the string.
Unquoted strings will have blanks and tab characters stripped off front and back when they are read, and may not contain commas, colons, quotes, // or embedded escape sequences (\", \n or \r). They may not start with a digit, sign, or decimal point (*i.e.*, must not be confused with a number). Colon, comma, end-of-line and comment (//) will terminate the un-quoted string.

(function) *New*. Examines the **DATA** statement(s) and checks for out-of-data. Can be used to avoid "Out of data" errors and determine what type of data is coming up.
Returns:
0 if at end-of-data,
1 if next data item is a string,
2 if next data item is a number.

The parameter is unimportant, so code: *type*=**READ** (0).

REM (statement) **REM** *comments*
Ignores the remainder if the line.
You may also use // instead of **REM**.

REMAINDER (function) **REMAINDER** *[no parameters]*
Returns the remainder following an **Integer-mode** LONGMATH division.
The remainder is saved, following the divide, so that you don't have to do the divide twice in order to get both the quotient and remainder.

There is no remainder for floating-point division.
See **FLOAT** and **INTEGER** statements.

RESTORE (statement) **RESTORE** [*statement-num*]
Allows **DATA** statements to be re-read from a specified line.
If statement-num is given, it must refer to a line containing a **DATA** statement. Otherwise, the first **DATA** statement in the program is used.
If there are more than one **DATA** statements on that line, it refers to the first one.

RETURN (statement) **RETURN** [*expression*]
Returns from a subroutine (see **GOSUB**).
expression is a numeric or string value to be returned from the subroutine. If no return value is supplied, zero is returned. This return value is used when the subroutine is called with a user-defined function (*var* = **FN***xxx* (…)) is called, and provides the value to assign to *var*. The return value is ignored for ordinary **GOSUB** calls.

RIGHT$** (function) **RIGHT$ (*x$*, *n*)
Returns the rightmost n characters of string *x$*.
x$ is a string expression.
n is an numeric expression that specifies the number of characters to be in the result.
If *n* is greater than or equal to **LEN** (*x$*), then *x$* is returned. If *n* is zero, the NULL string is returned.

RND (function) **RND** (*num_expr*)
Returns a [pseudo-] random number >= 0 and < 1.
Values returned are type DOUBLE.
The same sequence of "random" numbers is generated each time the program is run unless the random number generator is re-seeded (see **RANDOMIZE** statement).
If *num-expr* is positive, **RND** returns the next random number in the sequence.
If *num-expr* is zero, it repeats the last number generated.
If *num-expr* is negative, the random number generator is re-seeded. This is the same as calling **RANDOMIZE** with **ABS** (*num-expr*). Then the next random number is returned. | *num-expr* | must be >= 1 and <= 4294967295.
To get random integers in the range 0 through *n*, use **INT** (**RND** (1)*(*n*+1)).

ROOT (function) **ROOT** (*x* ,*y*)
Calculates the y^{th} root of *x*. Same as *x***^**(1/*y*), but has more precision for LONGMATH numbers. Calculated as **EXP** (**LOG**(*x*)/*y*). If *x* is negative, y must be an odd integer, e.g., **ROOT** (-8, 3) = -2. *y* cannot be zero unless *x* is also 0.

Notes:

ROOT (0, 0) = 1
ROOT (0, n) = 0
ROOT (*x*, 0) is an error.
ROOT (*neg-number*, *y*) is an error unless *y* is an *odd integer*.

RSET (statement) **RSET** *string-var* = *string-expression*

string-var is the name of a variable, normally one defined in a **FIELD** statement. (See **FIELD** statement). It may not be a string array, but it may be a string array element.

string-expression will be placed in the field (or "string variable") identified by *string-var*, typically in preparation for a **PUT** statement.

If *string-expression* is shorter than the width specified for *string-var* in the **FIELD** statement, **RSET** right-justifies the string in the field (spaces are used to pad the extra positions). If *string-expression* is longer than *string-var*, characters are dropped from the right.

Numeric (DOUBLE) values must be converted to strings before they are **RSET** (see **STR$** function). LONGMATH numbers shorter than 255 characters may be assigned to strings directly.

See "Working with Files" for more information on using RANDOM files.

While it is intended for use with field variables, **RSET** can also be used for ordinary string variables. **RSET** differs from an assignment statement in that the current length for the destination variable is preserved, and the string expression is truncated or padded to fit in that length. If the destination string variable has not been used yet, it will have a zero length and **RSET** will not place anything in it.

RUN (command) **RUN** [*file-spec*]

Begins the execution of a program.
RUN by itself runs the currently-loaded program, if possible.
RUN *file-spec* loads and runs the program.
file-spec (the file specification) must be a string. If the *file-spec* contains a colon, backslash(es) or embedded blanks, it must be in quotes – (see "Strings") You may use any file specification that works. If the path is in quotes, you must use a double backslash (\\) wherever a backslash is desired. (Don't use **..**\). If a complete path is not specified, the *file-spec* is appended to the **current working directory**.

Note: The **RUN** command is only valid from the command line. You cannot use it in a program. If you want your program to run another program, use the **CHAIN** statement instead.

Note: In GUI mode, it is faster to use the **RUN** button.

Note: The **RUN** button always loads a new copy of the program each time.

Note: If you have edited the program file, make sure you Save it before **RUN**ning again, or you will re-run the old file.

SELECT (function) **SELECT** (*message-string, array-name* [, *max-entri*es])

Displays a dialog box allowing you to select items from an array.
Returns the subscript (relative to 0) of the selected item, or -1 if no selection was made.
message-string is a string expression or constant to be displayed at the top of the dialog, *e.g.*, "Please select your preference…". It may contain up to 3 lines (separated by \n in the string).

array-name is the array containing the items from which you want to select.
The array must be a one-dimensional string array.
Just specify the name of the array – no subscripts.

max-entries [optional] is a numeric expression which limits the number of items displayed in the selection list, in case you don't want to use the entire array. The first non-null *max-entries* in the array will be displayed.

Note: If you double-click on an item, it will select that item and return immediately (no need to click the OK button).

Note: Only strings which are not null will be displayed.

Note: The contents of the array are **not** sorted before they are displayed. It is assumed that you have the array in the order you want the items to appear. If you want it sorted, first sort the array using the **SORT** statement.

SETCOLOR (function) **SETCOLOR** (*r, g, b*) Creates a color value for use with **SHAPE LINECOLOR, SHAPE FILLCOLOR, TEXT COLOR, PLOT COLOR, GRAPH BGCOLOR and GRAPH GRIDCOLOR.**

Example: *blue*=**SETCOLOR** (0, 0, 255)
TEXT COLOR=*blue*, …

This provides a simpler way of setting colors in graphics statements. You only have to define the color once, and can use it wherever a color specification is required, instead of having to code (*r, g, b*) in each statement.

You may also include the file **standard_colors.txt** or **basic_colors.txt**, which contain many pre-defined color values. (See the website sample programs page),

Note: Be sure ***not*** to use the color variable name somewhere else in your program as an ordinary variable.

(See "Intermediate Graphics" for more information)

SETDSTRULES (statement) **SETDSTRULES** *rules-array*

Sets a new Daylight Saving Time rule based on the values in the array.
The array must have 12 elements per row.
Changes remain in effect until a new BASIC program is run.
(See "*Working with Dates and Times*" for details)

SETDSTRULES "**ON**" | "**OFF**" [*quotes are required*]
"**ON**" allows DST rules to be applied (default),
"**OFF**" keeps everything in standard time.

SETTIMEZONE (statement) **SETTIMEZONE** *time-zone-bias*

Sets the current time zone (for DATETIME calculations).
The offset, *time-zone-bias,* is given in **minutes** and is negative for West of GMT, *e.g.,* Pacific Standard time is -8 * 60 = - 480 minutes.
Eastern time is -5 * 60 = -300 minutes.
Changes remain in effect until a new BASIC program is run.
(See "*Working with Dates and Times*" for details)

SGN (function) **SGN** (*num-expr*)

Returns the sign of a numeric expression.
Returns 1 if *num-expr* is positive, 0 if *num-expr* is zero, and –1 if *num-expr* is negative.

SHAPE (statement) **SHAPE** [*descriptive-parameter*][, …] [,*shape-drawing-parameter*] [, …]

Draws filled or open shapes on the graph.

Note: See the document "Intermediate Graphics" for more detailed information on the **SHAPE** stapement.

Descriptive Parameters:

Parameters which set the way figures are drawn:

AVERAGE = *num-expr* — Causes points in a shape to be averaged (move nearer to the value of adjacent points). 0 = no averaging, 10 = maximum.

LINEWEIGHT = *num-expr* — Thickness of outlining line (in points). Default is 1 point.

LINECOLOR = [(*r,g,b*) | *num-expr*]. Color of outlining line (default=black)

FILLTYPE = [SOLID | HATCH | NONE] — Specifies how the figure is to be filled.

FILLCOLOR = [(*r,g,b*) | *num-expr*] Color used for SOLID fills.

HATCH = [HORIZONTAL | VERTICAL | CROSS | RIGHT | LEFT | DIAGCROSS] — Hatch pattern used for HATCH fills.

SMOOTH = *num-expr* — Smoothes the figure using Bezier curves. 0 = no smoothing, 10 = maximum.

MOUSECLICK — Allows this shape to be selected by clicking on it (see "Advanced Graphics).

These set the size, location, angle, and/or endpoints of the figure:

LEFT = *num-expr* — Left side of bounding rectangle

RIGHT = *num-expr* — Right side of bounding rectangle

TOP = *num-expr* — Top of bounding rectangle

BOTTOM = *num-expr* — Bottom of bounding rectangle

CENTER = (*x,y)* — The center point (*x,y*) of the figure, or the center of the defining ellipse for **ARC**, **CHORD** and **PIE**.

RADIUS=*num-expr* — Radius of the circle (or defining circle for **ARC**, **CHORD** and **PIE**) or the distance from the center to any edge of a square. Forces the figure to be square or circular.

HRADIUS = *num-expr* — Sets the horizontal radius for an ellipse (or defining ellipse for **ARC**, **CHORD**, and **PIE**) or distance from

the center to the right or left edge of a rectangle.

VRADIUS = *num=expr* — Sets the vertical radius for an ellipse (or defining ellipse for **ARC**, **CHORD**, and **PIE**) or distance from the center to the top or bottom edge of a rectangle.

LINESTART=(*x, y*) — Specifies the starting point of a line.

LINEEND=(*x, y*) — Specifies the end point of a line.

CTLPT1=(*x, y*) — Specifies the first control point for a **BEZIER** curve.

CTLPT2=(*x, y*) — Specifies the second control point for a **BEZIER** curve.

STARTANGLE=*num-expr* — Specifies the starting angle for a **CHORD**, **ARC** or **PIE**. The figure is drawn counter-clockwise from the **STARTANGLE** to the **ENDANGLE**. Zero represents "directly to the right of the center point".

Note: Angles are given in radians (unless **DEGREES** is in effect) and must be between 0 and 360 degrees (0 and 2π radians).

ENDANGLE=*num-expr* — Specifies the ending angle for **ARC**, **CHORD** and **PIE**.

ROTATE=*num-expr* — Specifies the angle to rotate **TEXT** shapes.

ROTATECENTER=[(*x,y*) | "center"] — Specifies the coordinates of the center of rotation, or if "center", to rotate around the center of the figure.

POLYPOINTS=*array-name* — Specifies the name of the array containing the points for a polygon or polyline. The array must be dimensioned with **DIM** *array-name* (*n*,2), where *n* is greater than or equal to the number of points in the polygon or polyline. *array-name* (*n*,0) is the x-coordinate and array-name (*n*,1) is the y-coordinate. *n* is 0

for the first point. Don't specify the first point again at the end for a polygon.

The **POLYPOINTS** array is also used with **CHAROUTLINE** and **PLOTCHAROUTLINE**.

POLYCOUNT=*num-expr* Number of points in the polygon or polylines array (not counting the last point for polygons, which is the assumed to be the same as the start point).

CHARACTERS = *string-expr* String of characters to be used in a **SHAPE TEXT**.

FONT = *string-expr* Name of the font to be used for the **SHAPE TEXT** function. Default is a generic sans-serif font

BOLD = *num-expr* Weight (from 0 to 1000) of the font used in the **SHAPE TEXT** function.

ITALIC = *num-expr* Normal or italic for the font used in the **SHAPE TEXT** function. Default is 0 (= normal), 1 =itallic.

DIRECTION=[normal | reverse] Specifies the direction to draw figures: normal = counter-clockwise, reverse = clockwise.

RESET Resets all the descriptive parameters to their default or initial values.

Shape Drawing Parameters:

RECTANGLE Draws a rectangle (or square)

ELLIPSE Draws an ellipse (or circle)

ARC Draws an arc from **STARTANGLE** to **ENDANGLE**. The arc is not filled.

CHORD Draws a chord from **STARTANGLE** to **ENDANGLE**.

PIE	Draws a pie segment from **STARTANGLE** to **ENDANGLE**.
LINE	Draws a line from **LINESTART** to **LINEEND**.
POLYGON	Draws a polygon using the points specified in the **POLYPOINTS** array. The number of points in the array is specified in the **POLYCOUNT** parameter.
POLYLINE	Draws a polyline using the points specified in the **POLYPOINTS** array. The number of points in the array is specified in the **POLYCOUNT** parameter.
TEXT	Draws outline text shapes.
BEZIER	Draws a cubic Bezier curve from **LINESTART** to **LINEEND**, using the control points specified by **CTLPT1** and **CTLPT2** .
BEGINPATH	Begins a complex connected shape.
ENDPATH	Ends a complex connected shape.
CLOSEFIGURE	Ends one part of a complex connected shape, closing it up back to its starting point. Separates contours within the complex shape.
STROKE	Outlines a complex connected shape
STROKEFILL	Outlines and fills a complex connected shape.
CHAROUTLINE	Returns the outline of a font character in the **POLYPOINTS** array.
PLOTCHAROUTLINE	Plots the character outline in the **POLYPOINTS** array returned by **CHAROUTLINE**.

SHORT (function) **SHORT** (*num_expr*)

Returns a DOUBLE. If *num-expr* was LONGMATH it will convert it to a DOUBLE, if possible. If the number is out of range for DOUBLEs, an error will occor. Underflow will result in zero.

SIN (function) **SIN** (*num-expr*)
Returns the sine of an angle.
The angle is assumed to be in radians, unless the **DEGREES** statement is given. (see **DEGREES** and **RADIANS**)

SORT (statement) **SORT** *array-name* (name of the array to sort)
[(*index-1*, *index-2*, …)] (use indices to sort a sub-array)
[, count=*rows-to-sort*] (0 [default] = sort all rows)
[, keycol1=*primary-sort-column*]
[, keycol2=*seconday-sort-column*]
[, keycol3=*tertiary-sort-column*]
[, DESCENDING] (sort in reverse order)
[, CASE] (sort upper & lower case differently)
[, UNSTABLE] (default is stable)

See "The **SORT** statement and Sorting Arrays" in the "Advanced Features of QuickCalc" document for a detailed description of the SORT function.

SPACE$** (function) **SPACE$ (*num-expr*)
Returns a string of *num-expr* spaces.

SPAWN (statement) **SPAWN** *file-spec*$ [, *parameters*$ [, *directory*$]]
Begins the execution of a target file, document, shortcut, or website. This allows you to launch other applications from within your QuickCalc BASIC program. Use it carefully.
All 3 parameters are string expressions. The second two are optional.
The first parameter, *file-spec*$, must be a valid path to a document or file.
The second (optional) parameter, *parameters*$, is a string containing the parameters you wish to pass to the program.
The third parameter, *directory*$, is the default "working" directory for the program. If not specified, it will be the same as the current working directory for QuickCalc BASIC.

Note: If the parameters are quoted strings, make sure that all backslash (\ ) characters are represented by double-backslashes (\\ ).

Note: There is very little error reporting, other than syntax, for this function. Check your parameters carefully.

SQR (function) **SQR** (*num-expr*)
Returns the square root of *num-expr*. *num-expr* must be >= 0.

Note: for LONGMATH expressions, this is faster than *x* ^ (0.5) or **ROOT** (*x*, 2)

STARTTIMER (function) **STARTTIMER** *[no-parameters]*
Starts timing an interval.
Timing continues until an **ENDTIMER** function is called.
Returns 0 if timing is not supported, otherwise returns non-zero.

Note: Timing includes the time for interpreting the BASIC statements and all system overhead that occurs between the **STARTTIMER** and **ENDTIMER** function calls. For that reason, it is not an accurate indication of the time required to do mathematical functions.

STEP (see **FOR**).

STOP (statement) **STOP**
Works the same as the **STEP** button, that is, the program enters stepping mode (see "Debugging"). Can be used anywhere in the program. When the program is paused, you can enter most commands and statements. Typing <enter> alone causes the next statement to be executed, similar to clicking **STEP** (see "Debugging"). Typing **CONT** or clicking "**Continue**" will continue the program. Typing **END** will terminate it.

Note: **STOP** is the same as **DEBUG STEP**.

STR$** (function) **STR$ (*num-expr* [, *format-spec*])
Returns a string representation of the value in *num-expr*.
num-expr must be within the range of a DOUBLE, unless the special format string for LONGMATH numbers is used (see "Working With Long Numbers").
format-spec is an *optional second parameter*, which is an over-riding print format string (constant or expression) to be used with this statement only.
If *format-spec* is not given, and printformat$$ is not NULL, it will use that format.
If *format-spec* is not given, and printformat$$ is NULL, **STR$** will use "%23.16g" (16 digits, maximum) and strip off any leading blanks.

STRING$** (function) **STRING$ (*n*, *m*)
STRING$ (*n*, *string-expr*)
Returns a string of length n, whose characters all have ASCII code *m* or the first character of *string-expr*.
n and *m* are numeric expressions in the range of 0-255.

SWAP (statement) **SWAP** *variable-1, variable-2*

Exchanges the values of two variables. The two variables must be of the same type.
No additional storage is required for strings and LONGMATH variables, making it useful for sorting data.

TAB (function) **TAB** (*n*)

n is a numeric expression. Tabs to position *n*.
If the current print position is already beyond space *n*, **TAB** goes to position *n* on the next line.
If the **TAB** function is at the end of a list of data items, no carriage-return will be added, as if the **TAB** function had an implied semicolon after it.
Note: Tab is only valid for **PRINT [#]**[**USING**] .
It is not valid in **WRITE** statements.

TAN (function) **TAN** (*num-expr*)

Returns the tangent of the angle. The angle is assumed to be in radians, unless the **DEGREES** statement is given. (see **DEGREES** and **RADIANS**)

TEXT (statement) **TEXT STRING**=*string-var* [, *parameter* [, *parameter* [, …]]]

Places text or labels on your graph.

Required parameter:

STRING=*string-var*	contains text to display

Optional parameters are:

ANGLE=*string-expr*[1,2]	Rotate text counter-clockwise (0-2π radians or 0-360 degrees, if **DEGREES** is set.
BOLD	Use boldface type
BOXED[1,2]	draws a box around the text.
COLOR=[(*r, g, b*) | *num-expr*]	*r* = red color (0-255), *g* = green color, *b* = blue color
DATASIZE=*num-expr*[3]	font height in data units.
FONT=*string-expr*[4]	font to use for this and subsequent **TEXT** statements. Null string resets to default font.
H=*num-expr*[2]	percent (left-to-right) on graph.
ITALIC	Use italic type
JUSTIFY=[LEFT | CENTER | RIGHT] [1,2]	
NORMAL	Resets **BOLD** and **ITALIC**
SIZE=*num-expr*[3]	point size (6-100 for text, 10-30 for axis text)
V=*num-expr*[1]	percent (bottom-to-top) on graph.
X=*num-expr*[1,2]	x-coordinate (data point) for text.
XAXIS[2]	label the x-axis

X2=*num-expr*[5]	right side of bounding area for text.
Y=*num-expr*[1,2]	y-coordinate (data point) for text.
YAXIS[1]	label the y-axis
Y2=*num-expr*[5]	top side of bounding area for text.

[1] (not valid with **XAXIS**)
[2] (not valid with **YAXIS**)
[3] (specify either **SIZE** or **DATASIZE**)
[4] (**FONT** may appear without **STRING**)
[5] (**X** and **X2** define the bounding area for horizontal sliding text, **Y** and **Y2** define the bounding area for vertical sliding text)

(See the document "Intermediate Graphics" for more information)

THEN (see **IF**).

TIME$** (variable) **TIME$
Returns the time in a string: "hh:mm:ss"
The time is gotten once, when the QuickCalc program starts, updated with **UPDATEDATETIME**.

TO (see **FOR**).

TOLOWER$** (function) **TOLOWER$ (*string-expr*)
Returns a string equal to *string-expr* with all the upper-case letters (capital letters A - Z) converted to lower-case (a - z).

TOUPPER$** (function) **TOUPPER$ (*string-expr*)
Returns a string equal to *string-expr* with all the lower-case letters (a - z) converted to upper-case (capital letters A - Z).

UPDATEDATETIME (statement) **UPDATEDATETIME**
Updates the values in the "constants" **DATE$, DAYOFWEEK$,** and **TIME**$.
These constants will give the same values each time they are referenced, until **UPDATEDATETIME** is called. This allows you to print the same time on each page of a report. Updating the time lets you display start and stop times on a job or show the time each part of a long process finishes.

UPDATEDIALOG (statement) **UPDATEDIALOG** [*title* **;**] *variable-name-1*, *variable-name-2*, …
UPDATEDIALOG (function) rc = **UPDATEDIALOG** ([*title*,] *string-array*)

UPDATEDIALOG (statement or function) is exactly the same as **INPUTDIALOG**, except the edit fields **are pre-loaded with the current contents** of the target variables, saving you from having to re-type them and allowing you to change or edit them.

See the document "Advanced Features of QuickCalc" for complete information and examples of how to use **INPUTDIALOG and UPDATEDIALOG**.

UPDATEREGION (statement) **UPDATEREGION** *left, right, top, bottom*

This statement causes the selected region of the graph window to be refreshed. It is used when the text string referenced by a **TEXT** statement has changed. The graph is not automatically updated in this case.

Note: If you updated the entire screen when only a line of text had changed, you would notice a lot of flicker, particularly if the text changes often, like in a time display or counter. This statement addresses that problem.

left, right, top and *bottom* [numeric values in data units] define the rectangular region to update. The actual position of the region on the screen will be determined by the scale factors and offsets.

Only the area within the selected region will be updated, until the next time the entire graph is updated due to scrolling, resizing, auto-scaling, or plotting a new figure.

This statement will cause an error if a graph is not active.

USING (see **PRINT**).

VAL (function) **VAL** (*string-expr*)
Returns the numerical value of a string.
VAL strips blanks, tabs and line-feeds from the beginning of the argument string and ignores num-numeric characters following the number.
If the first character(s) of *sting-expr* are not numeric, **VAL** (*x$*) returns 0.
If the string is empty, **VAL** (*x$*) returns 0.

The result of **VAL** is a numeric string, which can be assigned to a numeric variable (DOUBLE or LONGMATH) or used in a numeric expression. If it is assigned to a string variable or printed directly, it will be treated as a string containing the numeric part of *x$*.

WEND (see **WHILE**).

WHILE (statement) **WHILE** *expression*
… loop statements …
WEND

If the expression is true (not zero or NULL), the loop statements execute until the **WEND** statement is encountered. Control is then returned to the **WHILE** statement and expression is checked again. If the expression is still true, the process is repeated. If it is not true, execution resumes with the statement following the **WEND** statement.

WHILE – WEND loops can be nested to any level. Each **WEND** matches the most recent **WHILE**.

You may not branch into or out of a **WHILE-WEND** loop. Statement numbers are checked to prevent this.

The while loop may begin and end anywhere in compound statements and may be contained within an **IF** clause.

WORKINGDIR$** (function) **WORKINGDIR$ *[no parameters]*

Returns a string containing the **current working directory**. This is the directory in which files are opened by default.

This string may be saved and used later in a **CHDIR** statement, if desired, or used to construct a path for opening files.

Unless you change it, this will be the directory specified in the shortcut that was used to load QuickCalc, or the current working directory in effect when QuickCalc was invoked from the (console-mode) command line.

WRITE (statement) **WRITE** *list-of-expressions* [;]

Works like **PRINT**, except for the following changes:

TAB( ) is not valid in **WRITE**. Tab positions are not used.

DOUBLE Numbers are printed showing up to 16 significant digits, and an exponent if necessary ("%22.16g").

DOUBLE Numbers do not have leading + sign and are not followed by a blank.

Items printed are separated by commas.

Strings are enclosed in quotes.

Number strings are not enclosed in quotes.

Embedded quote, c/r or l-f in strings are substituted by escape sequences (\", \r or \n).

LONGMATH values are printed in a special format (see "Number Representation and Conversion")

USING is not valid.

Note: you can mix **WRITE** and **PRINT** statements to the same line, *e.g.*, **PRINT** *a*, *b$*, *c*; **WRITE** *d*, *e$*, *f* (note the semicolon).

The purpose of the **WRITE** statement is to create a data file that can be read using **INPUT #**.

WRITE # (statement) **WRITE #** *file-num* ; *list-of-expressions* [;]

WRITEs to a file, random-access file buffer, or SERIAL port (*See* **WRITE** *and* **PRINT #***).*

If *file-num* refers to a **RANDOM** file, data is written to the buffer in preparation for a **PUT** to that file. **WRITE #** may not attempt to write beyond the end of the buffer. If the *list of expressions* does not end in a semicolon, the remainder of the buffer is padded with blanks. If you do a second **PRINT #** or **WRITE #** to the buffer, it will over-write the first, unless the first *list-of-expressions* ended in a semicolon. If you want to clear the buffer, just execute a **PRINT #** statement with no parameters.

Note: You cannot output a **LONGMATH** variable using **WRITE #** to a random-access file buffer. Use **PRINT #** instead. The length of the **LONGMATH** number must fit in the random-access file buffer.

If *file-num* refers to a SERIAL file, a complete message is written to the serial port. If the statement ends with a semi-colon, you must issue another **PRINT #** or **WRITE #** to complete the line. Writing beyond the buffer length will cause an error.

WRITE %

(statement) **WRITE %** *string-var*; [*list of expressions*][;]

Similar to the **WRITE** statement, except that the output goes to a string variable.

Some differences are:

If a line terminates with a semicolon, data will be placed in the destination string, but will also be retained in the buffer for subsequent **PRINT%** or **WRITE%**.

LONGMATH items may not be written with **WRITE%**. The way LONGMATH items are formatted for **WRITE** only gives you the length on the first line. Instead, assign the LONGMATH item to a string variable (if <= 255) or a string array. Then operate on the resulting string(s).

The maximum line length is 255 characters. If a data item will not fit on the remainder of the line, it will not be copied into the string, nor will any subsequent items in the list. The data will not wrap around or continue to the next line – (there is no next line). You should stay within the 255-character limit.

WRITE % is useful for formatting strings, particularly those that are going to be used later in reports or graph **TEXT** statements.

Note: **PRINT%** and **WRITE%** statements may be intermixed to create a string. Use the semicolon at the end of the list to show that the list is continued in the next statement. You may have to manually insert a comma between items in this case.

Note: There are other ways to format strings, including string concatenation (**+**), **MID$**, **LSET** and **RSET** with **FIELD** statements, **STR$**, **STRING$**, **SPACE$**, **FORMATDATETIME$**, **HEX$**, **HEXCONVERT**, **DATE$**, **TIME$**, etc. You may use all these tools to get the formatting you like, or stick with the functions with which you are most comfortable.