Working with Long Numbers.

Long Variables

You define a long variable with the **LONG** statement, which works similar to the **DIM** statement. You can define long variables and dimension long variable arrays in the same statement: **LONG** *a*, *b*, *c*(5,2), etc. Long names and arrays cannot duplicate short names and may not end with $.

Long (LONGMATH) variables may be used the same as short variables. The syntax is the same. A few limitations apply (see below).

Note: You can, optionally, choose to run your program with LONGMATH numbers and variables only. (See "Working With Long Numbers Only", below.)

Constants

Numeric constants entered into your program, typed on the command line, used in **DATA** statements, or input from the console or a file, may be either long or short. A number is presumed to be and treated as short unless:

- it is more than 16 (significant) digits long,
- it has an exponent out of the range of DOUBLE numbers (approximately +/- 308)

Numbers outside this range may only be assigned to LONGMATH variables. If they are used in an arithmetic operation, the operation will be performed using LONGMATH routines.

Numbers entered, short or long, are maintained as numeric strings until they are assigned to a variable or used in an arithmetic operation or function. They will then be converted to match the type of their destination variable or the required type for the function. If they are required to be short, they will be converted automatically. Note: this may cause an error if the exponent is too large.

Numbers can be forced to be long or short with the functions **LONG** (*x*) and **SHORT** (*x*).

LONGPI and **LONGE** represent long versions of the constants π and *e*. These values are calculated at the current floating-point length when they are 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, rounded to the desired length.

Integer vs. Floating-Point Numbers.

There is no difference between the two in terms of how the numbers are stored. Integer and Floating-Point refer to the way operations are performed on these numbers.

Integers are simply numbers with nothing to the right of the decimal point. All digits to the left of the decimal point are used in integer arithmetic. If the number was floating-point (i.e., had digits to the right of the decimal), the fractional digits will be ignored. Integer division always produces a quotient and a remainder. Trig functions, **EXP, LOG, SQR**, etc. are not valid in integer mode.

Floating-point numbers have a fixed number of total significant digits, and an exponent. Division is carried out to the specified floating-point length with no remainder. All answers are rounded to the current floating-point length.

Both formats are stored the same, and may be used interchangeably. A long integer with a lot of trailing zeros will be stored without those zeros, e.g., 2.5E+5000. Switching between integer and floating-point modes does not change numbers which have already been stored. Changing the floating-point length also does not affect existing numbers – only the results of subsequent calculations.

Range and Size of Long Numbers.

Long integers are limited to 1 million digits.

Floating point numbers are limited by the value you set for the floating point length (use the **FLOAT** statement). You may make them any size you like (up to 1 million digits) subject to the memory on your computer and your patience. Huge numbers can take a long time to calculate, even with the highly-optimized routines used in QuickCalc.

The range of long numbers is exponents up to ± 1,000,000. **Note:** the number 1E1000000 still takes only 4 bytes to store, but 1E1000000 – 1 is .9999—(1 million 9's)…. All floating point numbers are rounded to FLOATING_PT_LENGTH significant digits.

Arithmetic

Math functions are performed in the same sequence and priority as with regular numbers. Operations like **+**, **-**, *****, /, **%**, and **^** operate on long and regular numbers. If either argument is long, the other is converted to long and a long operation is performed, yielding a long result. If the resulting value is assigned to a short variable, it will be converted down (see "Conversion").

The sequence that operations appear in the program statement determines whether the operations are long or short. This can cause unexpected results if you code the statements incorrectly.

Examples:

2/3 => 0.66666666666666667 (short)
2/(**LONG**(3)) => 0.666666666666666666666666666666666666666… (long)

longvar = **SQR** (2) => 1.4142135623730951 (short, converted to long)
longvar = **SQR** (**LONG** (2)) =>
1.41421356237309504880168872424280785696…. (long)
longvar = 2: *longvar*=**SQR**(*longvar*)
=> 1.41421356237309504880168872424280785696…. (long)
2 is converted to long & assigned to *longvar*. Then a long **SQR** is done.
longvar = 88888888 * 1e13: **PRINT** *longvar*
result = 88888888000000007**0**000 (error caused by binary rounding in DOUBLE arithmetic.
longvar = **LONG** (88888888) * 1e13: **PRINT** *longvar*
result = 888888880000000000 (no error)

Calculating πr^2 depends on the order: (assume *r* is short)
LONGPI * *r* * *r* (does all multiplies in LONG)
r * *r* * **LONGPI** (does the first multiply as short).

More examples and a more detailed discussion are provided in the section "Number Conversion and Assignment"

These examples show how you may not get the precision you expected if you code it incorrectly. You can, of course, use parentheses to force the order of operations, and the **LONG** (*x*) and **SHORT** (*x*) functions can force conversion. The result of the calculation is converted (if possible) to match the destination variable.

If you wish to avoid the confusion and issues described above regarding which numbers are long and short, and how and when they are converted, you can choose to work with long numbers only (see "Working With Long Numbers Only", below).

Comparisons

You may mix long and short numbers or variables in comparison operations. The same rules apply. When a long and a short number are compared, the short is first converted to a long. The comparison may not be exactly as you expect (see the example above using 88888888 * 1e13.

Logic Operations

When used in a logic operation, long numbers are treated just like short ones. Zero=false, non-zero = true.

Functions

Special, highly-optimized functions are provided for **SQR, LOG, LOG10, EXP, EXP10, SIN, COS, TAN, ATN, ACOT, FACTORIAL, ROOT** and raise to power **(^).** These routines perform long arithmetic to the precision specified in the **FLOAT** statement. They are automatically called if the argument is long. Pay attention to the type or argument being passed. **SIN** (180 / **PI** * *angle*), where *angle* is short, gives a short result. Even if *angle* is long, 180/**PI** is a short calculation. For maximum precision, specify **SIN** (180/**LONGPI** * *angle*) or use the **DEGREES** statement with **SIN** (**LONG** (*angle*) (see notes, above, on arithmetic.)

These functions (with the exception of **FACTORIAL**) are not valid when you are in INTEGER mode.

Functions operating on long arguments return long results.

Functions which expect only short arguments will convert long to short, if possible.

Printing

Long numbers are printed as if they were long strings. They are printed to multiple lines of the "console" output. If you are printing to a file, each line will be as long as will fit in the file's record length.

Long numbers may be **formatted** with a **PRINT USING** statement. Use the format string "*www.pppc*", where

www is the total width of the output string (255 characters maximum).

decimal point (optional). If present, a decimal point is printed, even if there are no digits to the right of the decimal point.

ppp is the "precision" (optional), *i.e.*, the number of digits to the right of the decimal point. It may be from 0 to *www* – 3.

c (optional) means place a comma between every 3 digits left of the decimal.

Long numbers formatted in this way will have their decimal points **aligned** if they use the same format string. Example:

```
LONG a, b, c, d
a = LONGPI * 1e10
b = LONGE / 1e10
c = -2.5e7
d = 0
PRINT USING "40.20c", a
PRINT USING "40.20c", b
PRINT USING "40.20c", c
PRINT USING "40.20c", d
```

will print:

```
31,415,926,535.89793238462643383280
             0.00000000027182818285
  -25,000,000.00000000000000000000
             0.00000000000000000000
```

Note: Do not use this type of format string for DOUBLE variables.
Do ***not*** assign it to **PRINTFORMAT$$**.

Notes: Values printed this way will be rounded to the rightmost printed digit.
Values which are too large will be formatted in exponential notation.
Values which are too small will be printed as zero.
There will always be at least one digit or zero left of the decimal point.
When commas are used, you cannot print as many digits to the left of the decimal point, (obviously).
This type of format string can also be used with the **STR$** function.

See the section "Number Conversion and Assignment" for more, and for how long numbers are handled in the **WRITE** statement.

Converting between Long and Short.

(see the section "Number Conversion and Assignment")

Working With Long Numbers Only.

If you wish to avoid the confusion and issues described above regarding which numbers are long and short, and how and when they are converted, you can choose to work with long numbers only.

To initiate this mode, specify the statement **LONG** "**ALL**"
(The quotes around "**ALL**" are required.)

This is usually done at the start of the program. It affects all subsequent variables and calculations. The statement remains in effect until the end of the program, the start of the next program, or until you specify **LONG** "**ALL**", **OFF**.

When this mode is in effect,

all new variables are assumed to be LONGMATH,
all arrays dimensioned are created as LONGMATH
all constants entered or read are converted to LONGMATH,
all arithmetic is performed with long calculations,
all functions do LONGMATH calculations and return long results,
logic operations and comparisons are performed using long calculations,
PI and **EULER** (constants) return **LONGPI** and **LONGE** at the current floating-point length.

Things to keep in mind when using this mode:

Long (LONGMATH) calculations take more time than short (DOUBLE) calculations. Your program will run longer because things like indexes and counters (*e.g.*, in **FOR** statements) will all be using long calculations.
Long numbers must be formatted differently in **PRINT USING**, and **STR$** (see above).
You can use the **SHORT** function to convert a number to a DOUBLE, if required (e.g., **PRINT SHORT** (*x*)).
You can turn this mode off at any time using **LONG** "**ALL**" **OFF**.
There is a limit to the number of LONGMATH variables, and if all your variables are LONGMATH, you could exceed this number. (See "Maximum Number of Long Variables", below)

Maximum Number of Long Variables.

By default, you may have up to 100 long number (LONGMATH) variables. This includes array elements. Each long number requires 300 bytes plus storage dynamically allocated to hold the number itself.

You may change the number of LONGMATH variables with the "Customize" dialog.

Storage Requirements.

LONGMATH Numbers are stored in structures. The structure contains information about the number, and a buffer to hold the number if it is less then 512 digits. If the number is longer than 512, additional memory is dynamically allocated to hold the number. Each 4-byte word of memory holds 8 digits.

When a LONGMATH variable is defined, it is not assigned a structure until a value is assigned to it. Unused variables have a value of zero. You can allocate a large array of LONGMATH variables and none of them will require a structure until the first time each is assigned a value.

Long multiply and divide, plus power series functions like **LOG**, require additional buffers for temporary results. These buffers are allocated automatically as needed and freed when no longer needed. If any memory allocation fails, it will generate an error and cancel your program.

Speed

Long number (LONGMATH) operations take more time than short (DOUBLE) operations. Extremely long numbers can take a very long time for multiply, divide, **SQR** and power-series functions like **SIN, COS, TAN, COT, ATN, ACOT, LOG[10], EXP[10], LONGPI, LONGE, FACTORIAL, ROOT** and raise-to-power (**^**).

Although the LONGMATH routines are perhaps the fastest available, you should not set the floating point length longer than you need. You can operate with numbers up to 1 million digits long, if required. These routines are optimized to perform the above calculations in approximately one second (on a 3 GHz processor) at a length of 10000 digits. Calculation time is approximately proportional to the square of the number of digits, so if a calculation takes one second at 10000 digits, it will take about 100 seconds at 100000 digits, etc. **LONGPI** at 1 million digits takes about 1 hour and 40 minutes on a 3GHz processor.

You can use the **Kill Program** button to terminate a LONGMATH function that is taking too long. It will interrupt approximately 0.1 sec. after you click the button and will terminate the BASIC program without completing the calculation. The **Step** button will break into the program, but not until the BASIC statement terminates, which may be a long time.