



BASIC Scripting on RDAcIII Ver 1.30

The scripts are set up on the SD-Card in the /SCRIPT folder.

The SCRIPT.INI defines the Script name and Script execution frequency.

The 10 *.BAS files contains the BASIC program scripts. By default they are named SCRIPT1.BAS to SCRIPT10.BAS.

***NOTE!** The scripts can be named more appropriate names.*

```
/SCRIPT/SCRIPT.INI
```

```
/SCRIPT/SCRIPT1.BAS
```

```
....
```

```
/SCRIPT/SCRIPT10.BAS
```

The scripts are loaded into RDAcIII memory on a startup, and therefore require an RDAc reset when changes have been made to the scripts.

A script is a small basic program that will start executing from the first command line and exit when it is finished.

Scripts must never cause an infinite loop or fail to return to the system when they have finished executing.

The system supports from 1 to 10 scripts that can be set to execute at different time intervals based in the script configuration file.

SCRIPT.INI

This is the overall script configuration file and it tells the system which scripts to execute, and when they must be executed.

```
[SCRIPT01]
```

```
FileName=SCRIPT01
```

```
ScanTimeMs=0
```

```
ScanTimeSc=30
```

```
DayTimeMn=0
```

```
...
```

```
[SCRIPT10]
```

```
FileName=SCRIPT10
```

```
ScanTimeMs=0
```

```
ScanTimeSc=0
```

```
DayTimeMn=0
```

The main programming interface to the RDAc III system is via the Register Database, commonly abbreviated as RBase.

The new CSD/Tecmo defined commands to access the RBase from MINIBASIC are as follows:

RBGET(*type, address*) Get a data item from the RBase

RBPUT(*type, address, data*) Put a data item into the RBase

type: The data type to read or write

Rbase mnemonic	No	Data type	
RB_NONE	0	Invalid	
RB_DI	1	Digital Input	(1:xxxxx single bit)
RB_DQ	2	Digital output	(0:xxxxx single bit)
RB_INT	3	Signed 16 bit	(4:xxxxx single register)
RB_WORD	4	Unsigned 16 bit	(4:xxxxx single register)
RB_LONG_LH	5	Signed 32 bit	(4:xxxxx two registers)
RB_ULONG_LH	6	Unsigned 32 bit	(4:xxxxx two registers)
RB_FLOAT_LH	7	Real 32 bit	(4:xxxxx two registers)
RB_MFP32_LH	8	MODBUS MFP32	(4:xxxxx two registers)
RB_LONG_HL	9	Signed 32 bit	(4:xxxxx two registers)
RB_ULONG_HL	10	Unsigned 32 bit	(4:xxxxx two registers)
RB_FLOAT_HL	11	Real 32 bit	(4:xxxxx two registers)
RB_MFP32_HL	12	MODBUS MFP32	(4:xxxxx two registers)

12-255 Reserved

address: Based on the data type, the Rbase address.

For digital inputs (Type 1), address 1 would be the equivalent of the first digital input bit at 1:00001

For digital outputs (Type 2), address 1 would be the equivalent of the first digital output/coil bit at 0:00001

For INT, WORD, LONG, ULONG, FLOAT and MFP32 data types, the address is always in the 4:xxxxx area.

data: Based on the data type, the data to write

The following script will read a WORD value from the RBase register at 4:00021, multiply it by a constant 1.2345 and write the result as a REAL32 floating point format value to registers 4:00023 and 4:00024 in a first Low then High register order.

```
10 LET A = RBGET(4 , 21)
20 LET A = A * 1.2345
30 RBPUT(7 , 23 , A)
```

This previous example could also be written as the following single line of code, but it makes it more difficult to read.

```
10 RBPOT(7 , 23 , RBGET(4,21) * 1.2345)
```

Now we need to get to the core of Mini-Basic, the “LET” statement. Mini-Basic will evaluate arbitrarily complicated arithmetical expressions.

The operators allowed are the familiar ‘+’ ‘-’ ‘*’ and ‘/’, and also MOD (modulus). Use parentheses ‘(‘)’ to disambiguate the order of evaluation.

```
10 REM Read fahrenheit as a float from 4:03001
20 LET f = RBGET(7,3001)
30 LET c = (f - 32) / 1.8
40 REM Write celcius as a float to 4:03003
50 RBPOT(7,3003,c)
```

As well as these, there are a large number of mathematical functions built into Mini-Basic, for example POW(x,y), which does exponentiation, SQR(x) (square root), SIN(x), COS(x) and TAN(x), sine, cosine and tangent.

All the trigonometric functions take or return radians. To convert radians to degrees, divide by 2 * PI and multiply by 360.

The logarithm function, LN(x), takes a natural logarithm. To convert a natural, base e log to log10, divide by LN(10).

There are also two mathematical constants, PI and e (Euler’s number). Be careful not to use these as variable names.

Note that expressions such as
LET x = x + 1
are legal and are in fact very useful.

Variable names must be shorter than 31 characters, and must not duplicate any Mini-Basic keywords.

BRANCHING

Programs often need to make branching decisions. In Mini-Basic this is provided by the IF ... THEN statement.

```
10 REM Square root program.
20 LET x = RBGET(7 , 3001)
30 LET s = 0.0
40 REM Square root of negative is not allowed
50 IF x < 0 THEN 70
60 LET s = SQR(x)
70 RBPOT(7 , 3003 , s)
```

We have also introduced the REM statement. This simply adds comments to make the program easier for a human to understand. REM statements are ignored by the interpreter.

An IF ... THEN statement always takes a line number as an argument. This can be of the form IF x < y THEN b, as long as b holds a valid line number.

The operators that are recognized by the IF ... THEN statement follow: '=', '<>' (not equals), '>', '<', '>=', '<='.

We can also use AND and OR to build up complex tests

IF age >= 18 AND age < 65 THEN x

Use parentheses to disambiguate lengthy tests.

IF age >= 18 AND (age < 65 OR job\$ = "Caretaker")

The IF ... THEN operators can also be applied to string variables. In this case the strings are ordered alphabetically.

BOOLEAN OPERATORS

AND			OR			XOR		
A	B	Out	A	B	Out	A	B	Out
0	0	= 0	0	0	= 0	0	0	= 0
0	1	= 0	0	1	= 1	0	1	= 1
1	0	= 0	1	0	= 1	1	0	= 1
1	1	= 1	1	1	= 1	1	1	= 0

A simple script to read digital inputs 1:00253 and 1:00254 and turn on coil 0:00256 if they are both 1 (AND Gate):

```

10 LET A = 1
20 IF RBGET(1 , 253) AND RBGET(1 , 254) THEN 40
30 LET A = 0
40 RBPUT(2 , 256 , A)

```

A simple script to read digital inputs 1:00253 and 1:00254 and turn on coil 0:00256 if either one of them is 1 (OR Gate):

```

10 LET A = 1
20 IF RBGET(1 , 253) OR RBGET(1 , 254) THEN 40
30 LET A = 0
40 RBPUT(2 , 256 , A)

```

NOTE! *Inputs 253, 254, 255 and 256 are the four digital inputs on the RDAcIII base and Outputs 255 and 256 are the two relays, that can be directly wired in without any additional I/O devices.*

To enable Mini-Basic to compute complicated functions we need access to arbitrary amounts of memory. For this we have the DIM statement. It creates a special type of variable known as an array.

```
10 REM Calendar program.
20 DIM months$(12) = "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec"
30 PRINT "Day of birth?"
40 INPUT day
50 PRINT "Month?"
60 INPUT month
70 REM Make sure day and month are legal
80 IF day >= 1 AND day <= 31 AND month >= 1 AND month <= 12 THEN 110
90 PRINT "That's impossible"
100 GOTO 30
110 IF day <> INT(day) OR month <> INT(month) THEN 90
120 PRINT "Your birthday is", day, months$(month)
```

You might want to modify this program to contain another array, this time a numerical one, containing the lengths of the months. For the ambitious, you could also input the year, and check for February 29th. Arrays can have up to five dimensions. For instance you might want to hold a chessboard in a 2d array
DIM board(8,8)

It is possible to redimension arrays. For a 2d or higher array this effectively scrambles the contents, but one dimensional arrays are preserved.

It is possible to manipulate arrays of values just using IF ... THEN and GOTO, but it soon becomes very clumsy. For this reason Mini-Basic includes FOR NEXT loops.

Say we want to print out an array

```
10 REM Prints the days of the month
20 DIM months$(12) = "Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
30 FOR I = 1 TO 12
40 PRINT Month", I, months$(I)
50 NEXT I
60 PRINT "Done"
```

FOR loops can be nested,

```
10 FOR I = 1 TO 8
20 FOR J = 1 TO 8
30 PRINT I, J
40 NEXT J
50 NEXT I
```

It is also possible to provide a STEP value other than one.

```
FOR I = 100 TO 0 STEP -2
```

If you specify a null loop, such as FOR I = 10 TO 0, control will pass over the loop body and go to the next matching NEXT statement. The control variable must always be in the NEXT statement.

FOR ... TO loops can take complex expressions, such as FOR I = x TO x * x, in these cases the initial value, the end value, and the step value are calculated once and then never modified.

FOR ... NEXT loops can be nested up to 32 deep. If you attempt to jump out of a loop then you are likely to trigger errors. For instance this program will enter an arbitrary number of values into an Array

```
10 REM Median program.
20 LET N = 0
30 DIM array(N+1)
40 PRINT "Enter a number, q to quit"
50 INPUT line$
60 IF line$ = "q" THEN 100
70 LET N = N + 1
80 LET array(N) = VAL(line$)
90 GOTO 30
100 PRINT N, "numbers entered"
105 IF N = 0 THEN 1000
106 IF N = 1 THEN 210
110 REM Bubble sort the numbers
120 LET flag = 0
130 LET i = 1
140 IF array(i) <= array(i+1) THEN 190
150 LET flag = 1
160 LET temp = array(i)
170 LET array(i) = array(i+1)
180 LET array(i+1) = temp
190 LET i = i + 1
195 IF i < N THEN 140
200 IF flag = 1 THEN 120
210 REM print out the middle
220 IF N MOD 2 = 0 THEN 250
230 LET mid = array( (N + 1) / 2)
240 GOTO 270
250 LET mid = array(N/2) + array(N/2+1)
260 LET mid = mid/2
270 PRINT "Median", mid
1000 REM end
```

MATHEMATICAL FUNCTIONS:

ABS(x)	Absolute value (removes sign)
ACOS(x)	Angle Arc Cosine of X in radians
ASIN(x)	Angle Arc Sin of X in radians
ATAN(x)	Angle Arc Tangent of angle X in Radians
ATAN2(y,x)	Angle Arc Tangent where y and x are adj. and opp. sides
COS(x)	Cosine of angle X in radians
EXP(x)	Exponent
INT(x)	Integer value (removes fractional part)
LN(x)	Natural Logarithm (Base e)
LOG(x)	Logarithm
POW(x , y)	Power
RND(x)	Random value
SQR(x)	Square root of X
SIN(x)	Sine of angle X in radians
TAN(x)	Tangent of angle X in radians
FRAC(x)	Returns the fractional part of X

MATHEMATICAL CONSTANTS:

E	2.71281828	Euler's number
PI	3.14159265.	PI

STRING FUNCTIONS:

CHR\$	(x)
ERS\$	(x)
HEX\$	(x)
INKEY\$	()
LEFT\$	(target, length)
MID\$	(target, offset, length)
RIGHT\$	(target, length)
STR\$	(char, length)
STRING\$	(char, length)
TIMES\$	(x)
TRIM\$	(target)

STRING FUNCTIONS RETURNING NUMBERS:

ASC	()
INSTR	(target, seek, offset)
LEN	(target)
VAL	(target)
VALLEN	(target)

Alphabetical list of keywords

Each keyword in MiniBasic is listed alphabetically, with a brief description of its function and how to use it.

e – Euler's number.

The mathematical constant e , 2.71281828... The base of natural logarithms, and used in many formulae.

Usage

e

```
10 LET y = POW(e, -x)
```

ACOS – arc-cosine.

Calculates the inverse cosine of a number. The result is in radians. Input must be between -1.0 and 1.0 .

Usage

```
num = ACOS(numeric)
```

```
10 LET rad = ACOS(x)
```

AND – logical operator.

Used in IF ... THEN statements to perform two tests. If both are true then the test succeeds. Note it cannot be used as a bitwise AND operator, as in some programming languages.

Usage

```
IF relational AND relational THEN numeric
```

```
10 IF name$ = "Fred" AND age > 18 AND age <= 65 THEN 100
```

ASCII – get the numerical code for a character.

Calculates the computer's internal code for the first character in a string, or 0 if the empty string is passed. It is useful for performing direct manipulations on the representation, for instance testing for newlines (code 13).

Usage

```
num = ASCII( string )
```

```
LET x = ASCII("*abc")
```

x now contains the code for an asterisk, or 42.

ASIN – arc-sine.

Calculates the inverse sine of a number. The output is in radians. The input must be between -1.0 and 1.0 .

Usage

```
num = ASIN(numeric)
```

```
10 LET rad = ASIN(x)
```

ATAN – arc-tangent

Calculates the inverse tangent of a number. The output is in radians. Note that for very extreme values accuracy may be lost.

Usage

```
num = ATAN(numeric);
```

```
10 LET rad = ATAN(x)
```

CHR\$ - convert ASCII value to a string.

Converts the computer's internal numerical character code to a Mini-Basic string of one letter. It is useful for performing numerical manipulations with the code. For instance, to insert a newline call CHR\$(13).

Usage

```
str = CHR$(numeric)
```

```
10 LET X$ = "Line 1" + CHR$(13) + "Line2"
```

X\$ now contains "Line 1" and "Line 2" separated by a newline character.

COS – cosine

Calculates the cosine of an angle. The input must be in radians.

Usage

```
num = COS(numeric)
```

```
10 LET x = COS(degrees/180 * PI)
```

DIM – dimension an array.

Use DIM to create a named list of numbers or strings. This is extremely useful when dealing with large amounts of data. For instance, if you are writing a program for a company with several employees, you can DIM an array to hold all their names.

Arrays can have up to five dimensions. In practise, even on modern computers, memory fills up very fast with big arrays, and three dimensions is the maximum recommended.

There must be no space between the name of the dimensioned variable and the opening parenthesis.

```
10 DIM name$(100)
```

Creates an array of 100 names.

```
10 DIM map(width, height)
```

Creates a 2d array of width * height entries, maybe representing grid squares on a map.

Array elements are in the range 1 – maximum, so

```
20 LET map(1,10) = 2.0
```

sets the top left element to 2.0. map(width, height) is the bottom right element. If you try to access out-of-range elements the computer will throw an error.

MiniBasic allows you to resize an array at any point by calling DIM on it again. If the array is one-dimensional, elements will be preserved. If the array has higher dimensions then the elements will be scrambled. Resizing an array is useful if, say, you are inputting a list of employee names and don't know how many there will be. Arrays of zero dimensions may not be declared. MiniBasic also allows you to initialise arrays when you dimension them.

This

```
10 DIM days$[7] = "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun"
```

will declare an array of days of the week. This method is useful for defining data. For 2d arrays, the first dimension is the lowest (x)

dimension, so

```
DIM name$(2, 4) = "Fred", "Bloggs",
```

```
"Joe", "Sixpack"
```

```
"Homer", "Simpson"
```

```
"John", "Doe"
```

is the correct order.

Dimensioned variables are intimately connected with FOR ... NEXT

loops. Use the loop counter to index into your array.

Usage

```
DIM id(numeric, numeric)
```

```
10 DIM array(10)
```

Creates a single-dimensioned array of 10 numerical elements

```
10 DIM dictionary$(2, N)
```

Creates a 2-dimensional array of N * 2 strings

```
10 DIM factorial(10) = 1!, 2!, 3!, 4!, 5!, 6!, 7!, 8!, 9!, 10!
```

Creates a list of the first ten factorials

FOR - start a for loop

FOR ... NEXT loops are extremely useful in programming. The FOR statement consists of three parts, the initial set-up value, a TO value, and an optional STEP value.

```
10 DIM array(100)
```

```
20 FOR I = 1 TO 100
```

```
30 INPUT array(I)
```

```
40 NEXT I
```

will input a hundred values into the array. The variable in the NEXT statement must be the same as that in the matching FOR.

FOR loops may be nested to a maximum depth of 32.

```
10 DIM chess(8,8)
```

```
20 FOR I = 1 TO 8
```

```
30 FOR J = 1 TO 8
```

```
40 chess(j,i) = 1.0
```

```
50 NEXT J
```

```
60 NEXT I
```

The step value does not need to be 1, and may be negative. For instance

```
10 FOR I = 1 TO 10 STEP 2
```

```
20 PRINT I
```

```
30 NEXT I
```

```
40 FOR I = 10 TO 1 STEP -0.3
```

```
50 PRINT I
```

```
60 NEXT I
```

The initial, to, and step values are calculated once on entering the FOR loop, they are then constant.

```
10 LET x = 10
```

```
20 FOR I = 1 TO x STEP x/5
```

```
30 PRINT I
```

```
35 REM Next line has no effect
```

```
LET x = x + 1
```

```
50 NEXT I
```

In MiniBasic, if the TO value is lower than the initial value (or higher if the STEP value is negative) then the loop does not execute. Control passes to the first matching NEXT.

It is important not to jump out of FOR ... NEXT loops, or get the nesting order wrong, otherwise MiniBasic's control flow will become confused.

The "Too many FORs" error is likely to be caused by jumping out of a loop. To terminate a loop prematurely, set the counter to the TO value, and jump to the matching NEXT

Usage

```
FOR id = numeric TO numeric STEP numeric
```

```
...
```

```
NEXT id
```

FRAC – Return the fractional part of a real number.

Usage:

```
10 LET A = FRAC(X)
```

Where FRAC(123.456) will make A = 0.456

GOTO – unconditional jump

GOTO executes a jump to another line. The line number is usually a constant, but GOTO x is supported.

GOTO is not considered good programming practise, but is essential in MiniBasic because flow control is so simple.

Usage

GOTO numeric

```
10 GOTO 100
```

```
10 GOTO x
```

IF – conditional jump

The IF ... THEN construct allows a MiniBasic program to make decisions. It can emulate any other control structure.

If the test condition is true, then control jumps to the line indicated after the THEN keyword. If false, control passes to the next line. No statements other than a line number may appear after the THEN keyword, though the form

```
IF y < 10 THEN x
```

is supported

The relational operators are =, <> (not equal) , >, >=, < and <=. They can be applied to strings or to numerical expressions.

The AND and OR logical operators can also be used. MiniBasic does not perform lazy evaluation – all expressions will be evaluated so all array indices etc must be legal.

Usage

IF relational THEN numeric

```
10 IF x < 10 THEN 100
```

```
10 IF a$ <> "OK" AND a$ <> "YES" THEN x
```

INPUT – input a number or a string.

To input data, use the INPUT function. In a test environment this will usually be typed by the user, if MiniBasic is a component of another program input will be provided by caller.

```
INPUT x
```

inputs a variable. Any non-numerical characters are skipped over until a number appears.

```
INPUT n$
```

inputs a string. Characters are read up to the first newline, which is discarded.

Most consoles provide data a line at a time, so input you type will not be available until you press ENTER.

If the input stream comes to an end, the program will fail with an error message.

Usage

```
INPUT id
```

```
10 INPUT x
```

```
20 PRINT x
```

INSTR – in string

Looks for occurrences of a substring within a string. The first argument is the string to search, the second argument the string to search for, and the third argument the position at which to start. The return value is 0 if the string is not found, or else the offset of the first occurrence.

For instance

```
x = INSTR("zigzag", "zag", 1)
```

will return 4

It is very useful for string manipulation. For instance, to test if the onecharacter string ch is a digit we could write

```
10 IF INSTR("0123456789", ch, 1) <> 0 THEN 100
```

Usage

```
num = INSTR(string, string, numeric)
```

```
10 LET x = INSTR(sentence, "and", 1)
```

INT – convert real to integer.

All MiniBasic numerical variables are stored as floating point. INT()

returns the lower integer portion of the number. Thus INT(1.9) = 1.

To round, call INT(x + 0.5).

INT() is also useful for getting rid of small errors caused by floating-point calculation.

Usage

```
num = INT(numeric)
```

```
10 LET x = INT(x/2)
```

LEFT\$ - returns the left portion of a string.

To take the leftmost characters of a string, call LEFT\$. If the string is too short to contain that number of characters, it returns the whole string.

Usage

```
str = LEFT$(string, numeric)
```

```
10 LET hello$ = LEFT$("Hello World", 5)
```

LEN – returns the length of a string.

To find the length of a string in characters, call LEN(). The empty string "" returns 0. It is often necessary to examine each character of a string for processing.

```
5 REM PRINT A$, omitting the letter "x"
```

```
6 INPUT A$
```

```
10 FOR I = 1 TO LEN(A$)
```

```
20 LET ch$ = MID$(A$, I, 1)
```

```
30 IF ch$ = "x" THEN 50
```

```
40 PRINT ch$;
```

```
50 NEXT I
```

```
60 PRINT
```

Usage

```
num = LEN(string)
```

```
10 LET x = LEN("This is a string")
```

LET – assignment

The LET statement assigns a variable a value. If the variable does not exist it is created.

```
10 LET x = 10
```

```
10 LET name$ = fname$ + " " + sname$
```

Plain variables like x or length are always numerical, string variables like name\$ always end with a dollar sign. It is illegal to try to assign a variable of the wrong type.

LET will not create or increase the size of a dimensioned variable

```
10 DIM array(2,2)
```

```
15 REM Legal
```

```
20 LET array(1,2) = 10
```

```
25 REM Illegal out of bounds
```

```
30 LET array(1,3) = 0
```

The form

```
10 LET x = x + 1
```

is legal and is often very useful. It is even legal if x has not been created

(it is initialised to zero).

Usage

```
LET id = numeric
```

```
LET id$ = string
```

```
10 LET x = 10
```

```
10 LET x = x + 1
```

```
10 LET a$ = CHR$(13)
```

LN – natural logarithm

Computes the natural logarithm of a number, which must be greater than zero. Natural logarithms are to the base e.

To convert to a base 10 logarithm,

```
LET log10 = LN(x)/LN(10)
```

To convert to a base 2 logarithm

```
LET log2 = LN(x)/LN(2)
```

Usage

```
num = LN(numeric)
```

```
10 LET log = LN(x)
```

MID\$ - middle string function.

Use this function to obtain a string from the middle of another string.

The first argument is the target string, the second argument the offset (1 – based) and the third argument the length of the substring to extract.

```
LET x$ = MID$("Distraction", 4, 5)
```

sets x\$ to "tract"

If the length is too long for the target string, the answer is truncated.

By passing –1 for the length, we tell the function to extract the remainder of the string.

```
LET x$ = MID$("Distraction", 4, -1)
```

sets x\$ to "traction".

Usage

```
str – MID$(string, numeric, numeric)
```

```
10 LET x$ = MID$(y$, 3, 4)
```

```
10 LET x$ = MID$(y$, 3, -1)
```

MOD – modulus.

Modulus is not a function but an arithmetical operator. It calculates the remainder after division.

```
LET x = 12 MOD 5
```

sets x to 2.

Both sides of the MOD operator should be of the same sign. MOD 0 is an error.

MOD also works for fractional values. 0.75 MOD 0.5 equals 0.25

Usage

```
num = numeric MOD numeric
```

```
10 LET X = Y MOD 10
```

```
10 LET X = Y MOD -0.1
```

NEXT – terminates a FOR ... NEXT loop

For description see FOR

Usage

```
NEXT id
```

```
10 FOR I = TO 10
```

```
20 PRINT I
```

```
30 NEXT I
```

OR – logical operator.

Used in IF ... THEN statements to perform both tests. If either one is true,

then the test succeeds and the jump is taken.

```
IF job$ = "caretaker" OR age < 65 THEN 100
```

Note that lazy evaluation is not performed. Both sides of the expression will always be evaluated.

When used in conjunction with AND use parentheses to disambiguate.

Usage

```
IF relational OR relational THEN numeric
```

```
10 IF x = y OR x = z OR x < 0 THEN 100
```

PI – mathematical constant.

The mathematical constant PI, or 3.14159265... This is the ratio of a circle's circumference to its diameter and is used in many mathematical formulae

Usage

```
PI
```

```
10 LET area = radius * radius * PI
```

POW – exponentiation function.

POW() raises x to the power y. Fractional and negative powers are supported.

```
LET x = POW(10, 2)
```

Will set x to 100.

By passing 1/y as the exponent, we can obtain the yth root of x. For example, to obtain the cube root of two pass

```
LET x = POW(2, 1/3)
```

Passing a negative power calculates the reciprocal. For example

```
LET y = POW(x, -3)
```

sets y to 1/(x^3)

Some values are illegal. For instance, POW(-1,1/2) will produce an indefinite result.

Usage

```
num = POW(numeric, numeric)
```

```
10 LET a = POW(x,y)
```

PRINT – output statement

All of MiniBasic's output is via the PRINT statement. It is used to print both numbers and strings.

```
PRINT "Hello World"
```

Will output the string "Hello World", followed by a newline.

```
PRINT x
```

Will print the value of x in a human-readable format.

It is possible to print many values in one line by separating them with commas.

```
PRINT "Your salary is", x, "Mr" name$
```

The comma will automatically insert a space.

To suppress the newline, terminate the PRINT statement with a semicolon.

```
10 LET x = 10
```

```
20 LET y = 25
```

```
30 PRINT x;
```

```
40 PRINT y
```

will output the string "1025"

To print a bare newline, use the empty string

```
10 PRINT ""
```

Usage

```
PRINT numeric or string, numeric or string ; (optional)
```

```
10 PRINT "Hello World"
```

```
10 PRINT x
```

```
10 PRINT "Hello", name$
```

```
10 PRINT "Enter your telephone number";
10 PRINT ""
```

REM – remarks

This statement is purely for adding comments to programs so that a human reader can understand them. It is also frequently used for “commenting out” code – prefixing with a REM so it is not executed. MiniBasic allows for multi-line comments, as long as the first character of every continued line is a space.

Usage

REM any comments

```
10 REM Demonstration program by Malcolm McLean
10 REM This is an extremely long comment, which is spread
over two lines.
10 REM PRINT "This PRINT statement is commented out"
```

RIGHT\$ - get rightmost characters of a string.

This is the twin to LEFT\$. It takes the rightmost characters of a string.

For instance

```
LET A$ = RIGHT$("Beholden", 3)
```

Would set A\$ to “den”.

If the target string isn’t long enough, all of the string is copied.

Usage

str = RIGHT\$(string, numeric).

```
10 LET A$ = RIGHT$(B$, 10)
```

RND – Random number generator

Many applications need random numbers. RND() provides a pseudorandom number generator. The argument, which should be an integer, tells RND() to generate a random integer in the range 0 to N – 1

```
10 FOR I = 1 TO 100
```

```
20 PRINT RND(10)
```

```
30 NEXT I
```

will output a stream of random digits in the range 0 – 9.

If we pass RND() the value 1 the number generated is a floating point value in the range 0 – (slightly below) 1.

The random number generator is deterministic. To force a certain behaviour, call RND() with a negative argument. This will “seed” the random number generator.

```
LET dummy = RND(-10)
```

will give us numbers based from the seed 10

Calling RND(0) will always return 0.

Usage

num = RND(numeric)

```
10 LET die = RND(6) + 1
```

```
10 LET dummy = RND(-10)
```

```
10 LET p = RND(1)
```

SIN – sine

Returns the sine of a number. The argument must be in radians.

Usage

```
10 LET s = SIN(theta)
```

```
10 LET s = SIN(degrees/180 * PI)
```

SQR – square root

Calculates the square root of its argument, which must be positive.

Usage

```
num = SQR(numeric)
10 LET root2 = SQR(2)
10 LET dist = SQR( (x1-x2) * (x1-x2) + (y1 - y2) * (y1 - y2))
```

STEP – increment for a FOR loop.

STEP is by default 1, but can be any value, positive or negative. It is evaluated once when the FOR ... NEXT loop is entered. For further details see FOR

Usage

```
FOR id = numeric TO numeric STEP numeric
10 FOR i = 1 TO 100 STEP 10
10 FOR i = 100 TO 1 STEP -1
10 FOR i = min TO max STEP delta
```

STR\$ - convert numerical value to string.

The numerical value $x = 10$ and the string value x = "10"$ are two different things. To convert a number into a human-readable string use STR\$

Usage

```
str = STR$(numeric)
10 LET reg$ = "ABC" + STR$(num)
```

STRING\$ - tandem string.

If we need to create a string that consists of a shorter string duplicated many times, use STRING\$.

```
LET stars$ = STRING$("*", 20)
```

will create a string of twenty asterisks. A common use is creating variable numbers of spaces for output formatting.

Usage

```
str = STRING$(string, numeric)
10 PRINT STRING$(" ", 10), out$
```

TAN - tangent

Calculates the tangent of an angle, which must be in radians.

Usage

```
num = TAN(numeric)
10 LET x = TAN(theta)
10 LET x = TAN(degrees/180 * PI)
```

THEN – component of IF statement.

THEN introduces the jump destination which is taken if the expression in the IF statement is true. The expression is always evaluated, even if the branch is not taken.

For further details see IF

Usage

```
IF relational THEN numeric
```

TIMES\$() – Returns a string representing current time

Usage:

```
10 LET T$ = TIMES$(X)
```

```
X=1   T$ = "YYYY/MM/DD-HH:MM:SS"
X=2   T$ = "YYYY/MM/DD"
X=3   T$ = "HH:MM:SS"
```

TO – component for FOR ... NEXT loop

In a FOR ... NEXT loop, TO introduces the terminal value. When it is exceeded, the loop terminates at the next NEXT statement.

For further details see FOR

Usage

FOR id = numeric TO numeric

VAL – calculate the numerical value of a string.

This function converts a human-readable string containing numbers to a numerical variable. The string may contain numbers in scientific notation e.g. 1.5e20.

The string is read up until the first non-numerical character is encountered. If the string does start with a number, 0 is returned

Usage

```
num = VAL(string)
10 LET x = VAL("1024")
10 LET x = VAL("1.5e20")
```

VALLEN – length of value

This function is designed for use with VAL() to tell the caller how many numerical characters were translated. This is useful if stepping through a string containing many numbers. It also tells the caller whether a string is numerical or not – if non-numerical it returns 0.

```
10 INPUT a$
20 IF VALLEN(a$) <> 0 THEN 50
30 PRINT "You must enter a number"
40 GOTO 10
50 LET x= VAL(a$)
60 LET a$ = MID$(VALLEN(A$), -1)
```

This code will read a number from the input, and prompt is valid input is not entered.

Usage:

```
num = VALLEN(string)
10 LET slen = VALLEN("121 dalmations")
```

Errors

Sometimes MiniBasic will terminate with an error message. Usually these are due to typing mistakes or logic errors in the BASIC program.

Occasionally they may be caused by the computer running out of resources, by illegal input, or by internal errors in the MiniBasic interpreter.

Can't read program

You have called MiniBasic with something it cannot recognise as a MiniBasic program at all, for instance with a text file containing a nursery rhyme.

Program lines not in order

Lines have to be in numerical order. If lines are out of order, you will

receive this error.

Line not found

You have tried to jump to a non-existent line.

Syntax error line

This means that the computer has encountered a line it cannot understand. It is a catch all error, incorporating things such as identifiers starting with digits, or lines not terminated with a newline.

Out of memory

The computer has run out of memory. This may occur when you try to dimension a huge array, or it may occur at any time if the computer is low on resources, since MiniBasic uses memory internally. Be particularly careful when dimensioning arrays with variables.

Identifier too long

An identifier (variable name) is allowed to be only 31 characters long, including the \$ for a string identifier. For dimensioned variables the number is one less.

No such variable

You have attempted to use a variable that has not been initialised.

Bad subscript

You have tried to access a dimensioned array beyond its dimensioned size.

Too many dimensions

You have tried to dimension an array with more than five dimensions.

Too many initialisers

In initialising a dimensioned array, you have tried to list more values than you have space for.

Illegal type

You have tried to use a string variable as the counter for a for loop

Too many nested fors line

Maximum depth of FOR .. NEXT loops is 32. Exceeding this limit is probably due to problems with jumping out of FOR ... NEXT loops.

For without matching next

You have declared a FOR statement but not a matching NEXT

Next without matching for

You have declared a NEXT statement without a matching FOR

Divide by zero

You have attempted to divide by zero. This is a mathematical error

Negative logarithm

You have attempted to take the logarithm of zero or a negative number. This is a mathematical error.

Negative square root

You have tried to take the square root of a negative number. This is a mathematical error

Sine or Cosine out of range

You have attempted to pass a value not in the range -1.0 to 1.0 to the

ASIN() or ACOS() functions.

End of input file

An INPUT statement has encountered an end of file condition. This could be due to some problem with the computer's system.

Illegal offset

A string function has received an illegal value for a string offset, such as a negative second argument to LEFT\$()

Type mismatch

You have entered a string expression where MiniBasic was expecting a numeric expression, or a numeric expression where it was expecting a string.

Input too long

Input lines can be a maximum of 1023 characters long. Lines longer than this are almost certainly either errors or malicious attempts to exploit the system, so they are rejected.

Bad value

There has been an internal overflow. Usually this is caused by trying to calculate with ridiculously large value like 10.

Not an integer

A non-integer was used as an array index or to a function (like RND()) which naturally expects an integer. Note that floating point arithmetic is not exact so expressions like SQRT(3.0) * SQRT(3.0) may not return exactly 3.0. Use the INT() function to force a number to an exact integer.

ERROR

Unspecified error has occurred. This probably represents some internal problem.

DTIME(1) Returns time as a real value

TIME\$(x) Time String

X

1 YYYY/MM/DD-HH:MM:SS

2 YYYY/MM/DD

3 HH:MM:SS

FRAC(x) Returns the fractional portion of a floating point number