

Brassica BASIC

Mike Lee

August 11, 2022

Brassica interprets an expanded subset of 1975 Altair/Microsoft BASIC. This enables running programs of historical interest in their primæval state.

1 Lines

Line numbers label branching destinations, but are not strictly necessary elsewhere. Blank lines, numbered or not, are allowed. Lines beginning with # are comments. Colons separate multiple statements on the same line. The presence or absence of horizontal whitespace makes no difference anywhere outside of string literals, DATA values, and line numbers. Aside from the names of variables, the interpreter is entirely case-insensitive.

```
# infinite loop
100
110 let x = x + 1
    go to 100 : REM unnumbered line
```

2 Variables

Legitimate names consist of a letter followed by zero or more alphanumeric characters, possibly followed by a data-type indicator (% or \$), possibly followed by array subscripts. Names are case-sensitive, not limited to two characters, and may not contain reserved words (TO, INT, etc.). Arrays of character strings are supported. The six variables below are distinct and may coexist. Subscripts begin from zero. All BASIC variables are global in scope. Variables may be referenced without prior definition, in which case numbers are initialised to 0, and strings to "". Referencing an undimensioned array DIMs it as 0 – 10 on each subscript.

X	A numerical scalar.
X%	An integer-constrained scalar (signed).
X\$	A character string.
X(10)	A scalar element of a one-dimensional numerical array.
X%(4,6)	A scalar element of a two-dimensional integer array.
X\$(0,0,0)	One string of a three-dimensional array of strings.

3 Operators

In decreasing order of precedence:

(...)	Bracketing (grouping operation, including functions).
^	Exponentiation.
unary + -	Identity and negation.
* /	Multiplication and division.
+ -	Addition and subtraction.
= <> < <= > >=	Equal to, not equal to, less than, less than or equal to, greater than, greater than or equal to.
NOT	Bitwise logical NOT.
AND	Bitwise logical AND.
OR	Bitwise logical OR.
XOR	Bitwise logical XOR.

Operators of equal precedence, such as the six relationals, are applied in left-to-right order. The equivalence operator is a single `=`, which is also used for the assignment operator immediately after an expressed or implied `LET`. Logically false relations evaluate to 0, true relations to -1. Whitespace is ignored; be wary of `F OR` and `X OR`. Consecutive operators need not be separated by parentheses, but see below.

Relational operations are allowed between strings, and are case-insensitive (hence, `"a" = "A"`). Addition of strings performs concatenation. Arguments of logical operators are coerced to integers. `NOT(X%) = -(X% + 1)`.

The unary negation and logical NOT operators imply a bracketing extending to (not around) the next operator of lower precedence. The unary identity operator, `+`, acts only on the value to its immediate right. In effect, the unaries act as functions. Hence;

$$A^B^C = A^{+B^C} = (A^B)^C$$

while

$$A^{--B^C} = A^{-(-(B^C))} = A^{(B^C)}$$

and furthermore,

$$\begin{aligned} &A * NOT B + NOT C + NOT D + E AND F \\ &= A * NOT(B + NOT(C + NOT(D + E))) AND F \\ &= A * ((NOT B) - (NOT C) + (NOT D) - E) AND F \end{aligned}$$

4 Commands

All BASIC statements begin with a command keyword. The absence of an overt keyword implies LET.

CLEAR	Deletes all variables, arrays, and user-defined functions. May be followed by a positive number, which is ignored.
DATA	Lists constant values for READING. DATA 3, 4.5E-2, REM, A B C, " D,:E " REM two numbers and three strings
DEF FN	Defines a custom function. FN forms the beginning of its (case-sensitive) name, which must end with the appropriate return-type indicator (\$, %, or none). DEF FNA(X) = X + C DEF fnb\$(A\$,n) = MID\$(A\$,n,1) PRINT FNA(7) fnb\$("ABC",2)
DELAY	Suspends execution for a time. (Replaces delay loops.) DELAY 2 :REM waits two seconds
DIM	Specifies the domain of an array variable's subscripts. Each subscript runs from zero to its specified limit (with both extremes included). DIM A\$(5), X(2,6) REM X has 3 * 7 = 21 elements
END	Terminates execution quietly.
FOR...TO...STEP	Begins a loop, iterating over some (non-array) variable. The expression between FOR and TO is an implied LET, defining the iterator. The STEP (increment) is optional, and defaults to 1. The termination threshold, appearing after TO, is locked-in as a constant when the loop is initiated. X=3: FOR I=1 TO X: PRINT I: X=10: NEXT REM prints 1, 2, 3. Because the loop-termination condition is only tested at the bottom (by the NEXT), loops always execute at least once. FOR I = 9 TO 5 STEP +2: PRINT I: NEXT REM this prints 9. Beginning a new loop terminates any prior loop over the same iterator within the same subroutine (see GOSUB). Termination of a loop also terminates any loops nested within it. FOR I=1 TO 3: FOR J=1 TO 3: FOR I=1 TO 3 NEXT I: NEXT J REM next-without-for error at the 'NEXT J'

GOSUB	<p>Branches to a new subroutine. The destination must be a constant literal line number. While variables have global scope, loops are only visible within the subroutine they were initiated in.</p> <pre> 100 FOR I=1 TO 3: GOSUB 200 200 NEXT :REM next-without-for error here 10 REM this prints 1 2 5 6 8 20 FOR I=1 TO 8: PRINT I; 30 IF I=2 THEN GOSUB 50 40 NEXT I: PRINT: END 50 FOR I=5 TO 6: PRINT I;: NEXT I 60 RETURN </pre>
GOTO	<p>Branches to a line number. The destination must be a constant literal. The word 'GO' is reserved by BASIC in its own right.</p> <pre> GOTO 840 </pre>
IF...GOTO	<p>Conditional branching. If the condition is true, execution jumps to the specified line. Otherwise, execution continues with the next line. Non-zero numbers and non-empty strings are considered true.</p> <pre> IF X > 5 GOTO 1000 </pre>
IF...THEN	<p>Conditional execution. If the condition is true, execution continues along the same line. If the condition is false, execution moves to the next line. Non-zero numbers and non-empty strings are considered true.</p> <pre> IF X > 5 THEN 1000 :REM same as IF - GOTO 10 REM GOTO does the work of ELSE 20 IF A\$ THEN X=X+1: GOSUB 500: GOTO 40 30 X=0: B\$="Z" 40 REM line 30 is the ELSE block </pre>
INPUT	<p>Accepts input from the terminal. The user is automatically re-prompted on entering the wrong type or number of comma-separated values. (To enter a string with a comma in it, wrap it in double quotes.)</p> <pre> INPUT :REM just wait for enter INPUT X(4),Y\$:REM expects two values </pre> <p>An optional string-literal prompt can be printed. It must be followed by either a semicolon (append the usual question mark) or a comma (no question mark).</p> <pre> INPUT "Coordinates";X,Y INPUT "[press enter]", </pre>

LET Assigns a value to a variable. The keyword is optional. The data types of value and variable must match. Non-integer numbers are floored when necessary.

```
LET A$ = "Hi!" :REM assigns "Hi!" to A$
X = "A" = "a" :REM assigns -1 to X
N%(2.3) = 4.9 :REM assigns 4 to N%(2)
```

NEXT Bottom of a loop. Increments the iteration variable by the **STEP** defined when the loop was initiated. If the variable then exceeds the termination threshold, the loop terminates and execution continues onward. Otherwise, execution returns to the top of the loop. If the iteration variable is not stated, **NEXT** applies to the most recent loop. When the **STEP** is non-negative, ‘exceeds’ means ‘is greater than’. When the **STEP** is negative, ‘exceeds’ means ‘is less than’. Terminating a loop also terminates any nested loop. See **FOR** and **GOSUB**.

```
FOR I = 8 TO 3 STEP -2: NEXT
FOR J = -5 TO -2: FOR K = 1 TO 3: NEXT K,J
REM at this point, I = 1, J = -1, K = 4
```

ON . . . GOSUB Branches to the *n*th of a list of subroutines. If *n* should be zero, or exceed the number of subroutines, no branch is made, and execution continues with the next statement.

```
N = 2: ON N GOSUB 1000, 2000, 3000
REM branches to the subroutine at line 2000
```

ON . . . GOTO Branches to the *n*th of a list of destinations. If *n* should be zero, or exceed the number of destinations, no branch is made, and execution continues with the next statement.

```
ON INT(3*RND(1)+1) GOTO 500, 600, 700
REM goes to one of these three lines
```

PRINT Sends visible output to the terminal. Numbers are printed with a trailing space. Positive numbers also have a leading space (in lieu of a negative number's sign). (Use `STR$(X)` to suppress these.) The width of the terminal is divided into 'print zones' of 14 spaces each. Consider;

```
PRINT A;BTAB(16)CHR$(34)SPC(4)":",C D$;
```

The first semicolon separates A from B, so we don't get the value of AB. No semicolon is needed after B, since the reserved word 'TAB' cannot be part of a variable's name. `TAB(16)` moves the cursor to terminal-column 16, where `CHR$(34)` prints a double-quote. `SPC(4)` moves the cursor four more spaces to the right, where a colon is printed. The comma then moves the cursor to the beginning of the next print zone, where the value of `CD$` is printed. (Whitespace is ignored, including within keywords.) The final semicolon says *not* to print a newline at the end.

`TAB`, `SPC` and comma are rapid operations, in that they take essentially no time even when teletypewriter-effect options are active. They do not overprint existing text. This is in contrast to printing spaces, which does take time, and does overprint. Cursor positioning and text-wrapping will be inaccurate when special characters, such as a bell or tab, have been printed to the line.

READ Assigns the next `DATA` value to a variable. See `RESTORE`.
`READ X,Y$:REM read a number and a string`

REM A remark; the rest of the line is a comment.

RESTORE Allows `DATA` to be `READ` again.
`RESTORE :REM re-READ from the beginning`
`RESTORE 600 :REM re-READ data from line 600`

RETURN Returns from a subroutine (to the point of `GOSUB`).

STOP Terminates execution with a break message.

5 Functions

ABS(X)	Absolute value of X.
ASC(X\$)	ASCII code of the first character of X\$.
ATN(X)	Arctangent of X.
CHR\$(X)	The character with ASCII value X.
COS(X)	Cosine of X.
EXP(X)	Natural exponential function of X.
INT(X)	Greatest integer less than or equal to X (floor function).
LEFT\$(X\$,N)	The leftmost N characters of X\$.
LEN(X\$)	The length, in characters, of X\$.
LOG(X)	Natural logarithm of X.
MID\$(X\$,I,N)	An N-character substring of X\$, starting from the Ith (or all characters from the Ith onward, if N is omitted).
INSTR(N,X\$,Y\$)	The position of the first occurrence of string Y\$ within string X\$, not coming before the Nth character (or the first, if N is omitted). Returns 0 when X\$ is empty, or when Y\$ does not appear.
POS(1)	Current position of the cursor across the console (the leftmost column is numbered zero).
RND(X)	A variate from the standard uniform distribution. Use X = 1 for a new variate, or X = 0 for the previous one.
RIGHT\$(X\$,N)	The rightmost N characters of X\$.
SGN(X)	Sign (signum) function of X.
SIN(X)	Sine of X.
SPC(X)	Advances the cursor X spaces to the right (or left, if X is negative). Can only be used within PRINT statements.
SQR(X)	Square root of X.
STR\$(X)	Converts X to character (string) representation.
STRING\$(N,X\$)	Concatenates N copies of X\$ (or N spaces, should X\$ be omitted).
SYST(1)	Current system date-time, in seconds.
TAB(X)	Positions the cursor at column X (or -X spaces in from the right margin, if X is negative). Can only be used within PRINT statements.
TAN(X)	Tangent of X.
TTW(1)	Width of the terminal, in characters.
VAL(X\$)	Converts X\$ to its numerical value.

The dummy argument to POS, SYST, and TTW must have the value 1. POS and TAB will be inaccurate when special characters (\a, \b, \t, etc.) have been printed since the last carriage return.

6 Identities

Functions on the left are not implemented directly.
Substitute expressions from the right.

A MOD B	$A - \text{INT}(A/B) * B$
LOG10(X)	$\text{LOG}(X) / \text{LOG}(10)$
SEC(X)	$1 / \text{COS}(X)$
CSC(X)	$1 / \text{SIN}(X)$
COT(X)	$1 / \text{TAN}(X)$
ARCSIN(X)	$\text{ATN}(X / \text{SQR}(1 - X * X))$
ARCCOS(X)	$1.5708 - \text{ATN}(X / \text{SQR}(1 - X * X))$
ARCSEC(X)	$1.5708 * \text{SGN}((X) - 1) + \text{ATN}(\text{SQR}(X * X - 1))$
ARCCSC(X)	$1.5708 * (\text{SGN}(X) - 1) + \text{ATN}(1 / \text{SQR}(X * X - 1))$
ARCCOT(X)	$1.5708 - \text{ATN}(X)$
SINH(X)	$(\text{EXP}(X) - \text{EXP}(-X)) / 2$
COSH(X)	$(\text{EXP}(X) + \text{EXP}(-X)) / 2$
TANH(X)	$1 - 2 * \text{EXP}(-X) / (\text{EXP}(X) + \text{EXP}(-X))$
SECH(X)	$2 / (\text{EXP}(X) + \text{EXP}(-X))$
CSCH(X)	$2 / (\text{EXP}(X) - \text{EXP}(-X))$
COTH(X)	$1 + 2 * \text{EXP}(-X) / (\text{EXP}(X) - \text{EXP}(-X))$
ARSINH(X)	$\text{LOG}(X + \text{SQR}(X * X + 1))$
ARCOSH(X)	$\text{LOG}(X + \text{SQR}(X * X - 1))$
ARTANH(X)	$\text{LOG}((1 + X) / (1 - X)) / 2$
ARSECH(X)	$\text{LOG}((\text{SQR}(1 - X * X) + 1) / X)$
ARCSCH(X)	$\text{LOG}((\text{SQR}(1 + X * X) * \text{SGN}(X) + 1) / X)$
ARCOTH(X)	$\text{LOG}((X + 1) / (X - 1)) / 2$