**Variables**

Variable names can be any length, using the letters A-Z and the numbers 0-9, but must begin with a letter.

Numeric variables are all integer.

String variables use the standard postfix ‘$’ and can be up to 255 characters long.

Single dimension arrays of either are supported.

To create a single value, refer to it on the left-hand side of assignment (or FOR). It is not created as part of an expression.

Arrays can only be created using a DIM statement. Trying c$(4) without it will fail.

A-Z are fast variables stored in a fixed place in memory. They are always present and are not zeroed when the program starts. This is deliberate.

**Operators**

In precedence order

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operators | | | | | | Function |
| & | | | ^ |  |  |  | Bitwise AND OR and XOR of integers |
| < | > | = | <= | >= | <> | Comparison (integer or string), Return -1 or 0 |
| + | - |  |  |  |  | Arithmetic, strings can be added (concatenation) |
| \* | / | % | >> | << |  | Arithmetic and logical shift. |

**Tokenisation**

Tokenisation is a simple form of analysis of the structure of the program. This matters with respect to spaces.

When trying to find if the word typed is PRINT, there are two ways of doing it. The first is when PRINT appears in the text as is; the second is when a complete identifier is the word PRINT.

So, if you have the text printa=1. This can mean two different things.

Under the first it means PRINT A=1 e.g. it prints -1 if A is 1, 0 otherwise.

Under the second it means PRINTA = 1 e.g. it assigns 1 to a variable called PRINTA

At present we use the second. This is because the first disqualifies all the variables beginning with a keyword (TOtal for example, it begins with “TO”). This means though it’s a requirement to put spaces between certain keywords. You cannot write just proccalculate as it thinks that’s one thing; you must put in proc calculate.

**Unary functions**

|  |  |
| --- | --- |
| Function | Purpose |
| abs(x) | Returns |x| |
| asc(a$) | Returns ASCII code of first char of a$ |
| chr$(n) | ASCII code to 1-character string |
| inkey() | ASCII code of current key pressed, or 0. |
| instr(s1$,s2$) | Find position of s2$ in s1$ ; returns 0 if not found. |
| left$(a$,n) right$(a$,n) mid$(a$,n) mid$(a$,n,m) | String split up commands. |
| peek(a) deek(a) leek(a) | Byte, Word, Long memory reading. |
| rnd() | Random integer in range -MAXINT ... MAXINT |
| sgn(a) | Returns 0 if a = 0, -1 if a <0 1 if a > 0 |
| spc(n) | Returns n size string of spaces |
| str$(n) or str$(n,base) | Converts integer to string |
| upper$(a$) lower$(a$) | Case conversion. |

**Commands (to date, e.g. these work)**

Commands can be separated by colons and have line numbers. Every line has a number, but they aren’t necessary.

|  |  |
| --- | --- |
| Command | Purpose |
| [let] <var> = <expression> | Assign a value to a variable. LET is optional. |
| assert <expression> | Causes error if expression is false (e.g. assert count=22) |
| clear | Clear variables and stacks. |
| cls | Clear Screen |
| defproc <name>:endproc | Defines a procedure (there is no DEF FN) |
| dim <name>(size),<name>(size) | Dimension arrays, only one dimension at present. |
| end | Ends program. Also exits emulator. If you want to see results use STOP. |
| for <var> = <from> to <to> [step <s>] …. next [<var>] | Standard For Loop |
| gosub <line>/return | Standard GOSUB/RETURN. |
| goto <line> | Standard GOTO |
| if <expr>  [else]  endif | If then else that can spread over several lines |
| if <expr> then <commands> | One line test (no ELSE) |
| link <addr> | Call routine at <addr>. The fast variables A,X and Y are loaded into the processor registers before calling and saved on exit. |
| list [<from>][,<to>] | List program |
| new | Erase program |
| on <expr> goto l1,l2,l3,l4 | Selector, l1 is for 1, l2 for 2 etc. |
| poke a,b doke a,b loke a,b | Write byte/word/long to memory (note, there is no protection at all here) |
| print <expression>’;, | Print things. ‘ is a new line and , does a tab |
| proc <name> | Invokes a procedure. |
| rem “comment” | Comment : *MUST* be in quotes. |
| repeat: ….. : until <expr> | Repeat Loop |
| run | Run the program / Clear variables & stacks etc. |
| stop | Stop the program, also allows viewing of results in emulator. |
| vdu <expr>[;,]<expr>… | Output characters to console as numbers. |
| while <expr>: …. : wend | While loop |

**Running**

To run it, edit the basic.bas file in the develop directory and run dev.bat

Each time you run dev.bat it imports the text file in and runs the interpreter. When you exit the emulator with the ESC key it updates that basic.bas file and writes another copy of the source, time stamped, in the archive directory.

The purpose of this is to allow me (or anyone else) to try to write code without losing it if the interpreter breaks at some point, which it almost certainly will.

**Editing**

The front end, which is separate from the interpreter, is a rough mock-up of the CBM version, e.g. you can edit lines by just moving over them and pressing RETURN. Keys are Ctrl+L clear screen, Ctrl+WASD / TAB moves cursor.

The line numbers in basic.bas should be in order, even though this BASIC doesn’t need them.

The benchmarks for speed, based on the very retro PCW Benchmarks are in the Benchmarks file. The interpreter runs at about the same speed as Acorn Archimedes 3010 BASIC *if* the estimate I found of 3.1 MIPS for a 14Mhz 65816 is correct. I haven’t cycle counted it, just instruction counted it, so it may be different in practice.

The Ctrl+C key combination. functions as a “Break” key to stop programs running, or interrupt lists, as ESC is used to exit the emulator entirely.

Please report all bugs to [paul@robsons.org.uk](mailto:paul@robsons.org.uk) or put them on the git tracker <https://github.com/paulscottrobson/Basic65816>