Tokenisation

Tokenisation of BASIC code is fairly straightforward in some ways and more complex in others. Outside quoted strings, case is irrelevant.

Punctuation and Digits

Punctuation occupies 00-3F and is summarised in the associated spreadsheet. There are three different groups.

1. Normal characters with ASCII codes 33-63 which are used unconverted.
2. Characters from 64-127 are used by anding their lower 3 bits, moving bit 5 into bit 3 and adding these $10, mapping them all onto $10-$1F, and this is easily reversible.
3. Double characters, these are > or < followed by < = or > ; this gives 5 legal 2 character punctuation elements ; comparisons (<= >= <>) and shifts (<< and >>)

Identifiers

Identifier tokens are all 2 bytes long and range from $4000-$7FFF, with the high byte *first*.

The upper 14 bits (e.g. 0000-3FFF) are an offset into a variable area which is created as program lines are edited. All identifiers go in here, with their type information (int, string float, array types and procedure).

These are created as required during program editing, and identifiers cannot be deleted except using NEW (which is irreversible). CLEAR simply resets all the variables to zero or empty string ; they still exist. Array variables lose their data however, until they are re-dimensioned, though they still exist as entities.

Differentiation is done by postfix , # (float) $ (string) ((procedure or array). The latter are differentiated by being prefixed by the PROC keyword on the line. This is done in a scan at the start which first clears all procedures $18 back to integer arrays $04, then scans each line in turn looking for PROC <identifier>(

A consequence of this is that arrays and procedures cannot have the same name.

Identifier Record

Offset	Name	Description
0	Offset	Offset to next variable, 0 = list end
1	Hash	Hash of all the identifier characters (0-9 A-Z and _) and the type characters.
2	Type	$00 (integer) $08 (float) $10 (string) $18 (procedure) + $04 if array.
3	Data	(5 bytes, if a number, this is mantissa0..mantissa 3, exponent, otherwise low/high of address followed by size#1 and size#2 if an array. Only 2D arrays are allowed, up to 255.
8+	Name	Name starts here, stored in upper case. The last character has bit 7 set.

The type closely maps to the type byte on the number stack viz.

Bit	Purpose
7	Value is negative
6	Value is zero
5	Value is a reference address
4,3	Type (00 int,01 float, 10 string,11 proc)
2	Array
0..1	Reference size -1 (e.g. 1 byte ..4 long)

Keywords

The remaining tokens are in $80-$FF as is normal. There are special case tokens for quoted strings, which are stored as $FF <length of string> <characters> $00 and end of line ($80)

These are *all* characters A-Z only, so identifiers (A-Z and _, 0-9 possibly following) are all stored as identifiers or keywords. A slight optimisation can be made because keywords cannot contain 0-9 or _, so can be discounted when scanning.

Identifiers once identified as such are typed by seeing if they are followed by # or $ and subequently [, or if they are followed by (only.

Program

Program is stored in the preferred format

- A 1 byte offset to the next line ($00 means end of program, $01 means next bank)
- A 2 byte low/high line number. These are for editing, though GOTO GOSUB and RETURN are available for backward compatibility
- The tokenised line
- $80

Token groups

The first group is control structures ; these have a depth offset table associated with them (e.g. repeat is +1, until is -1)

The second group is unary functions. Because of the tokenising process for identifiers, checks are done on the textual (and _) part of the identifier only. So the token for left$ is left and the $ and (are checked manually.

Some unary functions are handled seperately. When evaluating a term, once it has been decided it is a “punctuation” (e.g. token is $00-$3F), but *not* a decimal digit, there are several unary functions to be handled, which are listed in the punctuation spreadsheet.