Appendix A — Syntax definition

This syntax definition is written in Backus-Naur form, or BNF, in a similar manner to the 'Syntax' sections in Chapter 33 of the *BBC User Guide*, or chapter 25 of the Electron *User Guide*. As well as the syntax of the keywords, it also includes the expression evaluator, and non-keyword statements. Although this syntax definition is not particualrly easy to read at first, it is very useful when trying to understand what BASIC is doing whilst decoding a particular statement or function.

Note that EVAL and FN may be either string or numeric functions (i.e. they may return either a string or numeric value).

OSCLI and OPENUP are not implemented in BASIC1

Symbols

The following symbols have special meaning in this section:

<> enclose defined items ('syntactic entities'), like **<numeric>** or **<factor>**.

::= should be read as 'is defined as'.

| should be read as 'or': it is used to separate alternative items.

denote possible repetition of the enclosed section zero or more times.

[] enclose optional items.

Any other symbols are as read (like '+' and 'MOD'). Note that the '<' and '>' symbols in the definition of **<relation operator>** do not enclose a syntactic entity, but are 'less than' and 'greater than' symbols respectively.

Example

As an illustration, the definition of the RENUMBER command is:

```
<renumber command> ::= RENUMBER [<line-num> [,<line-num>]]
```

There are two optional sections in this line, so the command can be one of three forms:

1) **RENUMBER**

2) **RENUMBER <line-num>** (e.g. **RENUMBER 1000**)

3) **RENUMBER <line-num>,<line-num>** (e.g. **RENUMBER 100,5** − the second number is not an actual line number, but syntactically it is just the same)

Statements

```
<immediate-statement> ::= <line-entry>
        | <command> | <statement>

<line-entry> : := <line-num><line>

<line> ::= {anything }{return }

<command> ::= {statement starting with a command keyword}

<statement> ::= <keyword-statement> | <assignment-statement>
        | <FN-return--statement> | <OS-statement>
        | <enter-assembter-statement> | <empty-statement>

<keyword-statement> ::= {statement starting with a key  ord}

<assignment--statement> ::= <num-var>=<numeric>
        | <string-var>=<string>

<FN-return-statement> ::= =<string> | =<numeric>

<Os-statement> ::= *<Line>

<enter-assembter-statement> ::= [

<empty-statement> ::= {nothing }

<auto command> ::= AUTO [<line-num> [,<line-num>]]

<delete command> ::= DELETE <line-num>, <line-num>

<Load command> ::= LOAD <string>
```

```
<List command> ::= LIST <line-num> | [<line-num>],[<L ine-num>]

<Listo command> ::= LISTO <numeric>

<new command> ::= NEW

<oLd command> ::= OLD

<renumber command> ::= RENUMBER [<line-num> [,<line-num>]]

<save command> ::= SAVE <string>

<ptr statement> ::= PTR# <factor>=<numeric>

<page statement> ::= PAGE =<numeric>

<time statement> ::= T|ME =<numeric>

<lomem statement> ::= LOMEM =<numeric>

<himem statement> ::= H|MEM =<numeric>

<bput statement> ::= BPUT# <factor>, <numeric>

<caLL statement> ::= CALL <numeric> {nothing ,<variabte>}

<chain statement> ::= CHA|N <string>

<clear statement> ::= CLEAR

<close statement> ::= CLOSE# <factor>

<clg statement> ::= CLG

<cls statement> ::= CLS

<colour statement> ::= COLOUR <numeric>

<data statement> ::= DATA <line>

<def fn statement> ::= DEF FN<variable name> [(<variable>
       {,<variable>})]

<def proc statement> ::= DEF PROC<variable name>
       [(<variable> {,<variable>})]

<dim statement> ::= DIM <dim section> {nothing ,<dim section>}

<dim section> ::=<variable>(<numeric> {nothing ,<numeric>})
       | <num-var><numeri c>

<draw statement> ::= DRAW <numeric>, <numeric>

<end statement> ::= END
```

```
<endproc statement> ::= ENDPROC

<envetope statement> ::= ENVELOPE <numeric>, <numeric>,
       <numeric>, <numeric>, <numeric>, <numeric>,
       <numeric>, <numeric>, <numeric>, <numeric>,
       <numeric>, <numeric>, <numeric>, <numeric>

<for statement> ::= FOR <num-var>=<numeric> TO <numeric>
       [STEP<numeric>]

<gcot statement> ::= GCOL <numeric>, <numeric>

<gosub statement> ::= GOSUB <numeric>

<goto statement> ::= GOTO <numeric>

<if statement> ::= IF <testabte-condition> [THEN<statement>
       | THEN<line-num>] {<statement> } [ELSE{<statement> }]

<input statement> ::= INPUT [LINE] {{ [<input-message>] ,|;}
       <variable>}

<input message> ::= <string-const> | <format-items>

<input# statement> ::= |NPUT# <factor> {{ ,<variabl-e>}

<let statement> ::= LET<string-var>=<string> |
       LET<num-var>=<numeric>

<local statement> ::= LOCAL {<variabte>}

<mode statement> ::= MODE <numeric>

<move statement> ::= MOVE <numeric>, <numeric>

<next statement> ::= NEXT [<num-var>]

<on-error statement> ::= ON ERROR <statement>|OFF

<on statement> ::= ON <numeric> GOTO|GOSUB <numeric>
       {<variabte> ,<numeric>} [ELSE <statement>]

<oscli statement> ::= OSCLI <string-factor>

<plot statement> ::= PLOT <numeric>, <numeric>, <numeric>

<print statement> ::= PRINT {" | , | ; | <format items> |
       <numeric> | <string>

<format items> ::= ' | SPC<factor> | TAB(<numeri c>[,<numeric>])

<proc statement> ::= PROC <variabte name> [(<variabte>
       { | , | ; | <format items> | ,<variable>})]
```

```
<read statement> ::= READ {[ ,]}

<rem statement> ::= REM<line>

<repeat statement> ::= REPEAT

<report statement> ::= REPORT

<restore statement> ::= RESTORE

<return statement> ::= RETURN

<run statement> ::= RUN

<sound statement> ::= SOUND <numeric>, <numeric>, <numeric>,
       <numeric>

<stop statement> ::= STOP

<trace statement> ::= TRACE ONlOFFl<numeric>

<until statement> ::= UNT|L <testable condition>

<vdu statement> ::= VDU <numeric> {[ ,|; <numeric>} [,|;]

<width statement> ::= W|DTH <numeric>
```

Expression evaluator

```
<numeric> ::= <testabte-condition>

<testable-condition> ::= <Logical-expression>
       {<logicat-expression> }

<Logical-expression> ::= <relnl-expression>
       {<retnt-expression> }

<relnt-expression> ::= <expression> |
<expression><reLati on-operator><expression> |
<string><reLati on-operator><string>

<retation operator> : := = | < | <= | <> | > | >=

<expression> ::= <term> {+ 1- <term>}

<term> ::= <sub-term> {<term-operator><sub-term> }

<term-operator> ::= * | / | MOD | DIV

<sub-term> ::= <factor> {^<factor>}

<factor> : := <primi tive> | —<primi tive> | +<primitive>

<primitive> ::= <function> | <num-var> | <num-const> |
```

```
       &<hex-number> | (<testabLe expression>)

<variable> : := <stri ng-var> | <num-var>

<num-var> ::= <simpLe-var> | ?<factor> | !<factor> |
       <simple-var>? <factor> | <simple-var> ! <factor>

<string> ::= <string-factor> {+ <string-factor>}

<string-factor> ::= <string-function> | <string-var> |
       <string-const> | (<string>)

<string-var> ::= <dynamic-string> | $<factor>

<num-const> ::= {number like 12 or 1.3E-15}

<line-num> ::= {positive decimal integer}

<hex-number> ::= {hexadecimal number like FFE4}

<simple-var> ::= {numeric variable like A% or FRED(3)}

<dynamic-string> ::= {string variabte like A$ or BBC$(1)}

<string-const> ::= {string in quotes, like "this string"}
```

Functions

```
<function> ::= {numeric-vatued function}

<string-function> ::= {string-val-ued function}

<abs function> ::= ABS<factor>

<acs function> ::= ACS<factor>

<adval function> ::= ADVAL<factor>

<asc function> ::= ASC<string>

<asn function> ::= ASN<factor>

<atn function> ::= ATN<factor>

<bget function> ::= BGET#<factor>

<cos function> ::= COS<factor>

<count function> ::= COUNT

<deg function> ::= DEG<factor>

<eof function> ::= EOF#<factor>
```

```
<erl function> ::= ERL

<err function> ::= ERR

<eval. function> ::= EVAL<string-factor>

<exp function> ::= EXP<factor>

<ext function> ::= EXT#<factor>

<fatse function> ::= FALSE

<fn function> ::= FN<variable name> [(<variable>
       {string-val-ued ,<variable>})]

<get function> ::= GET

<himem function> ::= H|MEM

<inkey function> ::= INKEY<factor>

<instr function> ::= INSTR(<string>, <string> [,<numeric>])

<int function> ::= INT<factor>

<lLen function> ::= LEN<string-factor>

<ln functi on> : := LN<factor>

<log function> ::= LOG<factor>

<lomem function> ::= LOMEM

<not function> ::= NOT<factor>

<openin function> ::= OPENIN<string-factor>

<openout function> ::= OPENOUT<string-factor>

<openup function> ::= OPENUP<string-factor>

<page function> ::= PAGE

<pi function> ::= PI

<point function> ::= POINT(<numeric>, <numeric>)

<pos function> ::= POS

<ptr function> ::= PTR#<factor>

<rad function> ::= RAD<factor>

<rnd function> ::= RND[(<numeric>)]
```

```
<sgn function> ::= SGN<factor>

<sin function> ::= S|N<factor>

<sqr function> ::= SQR<factor>

<tan function> ::= TAN<factor>

<time function> ::= T|ME

<top function> ::= TOP

<true function> ::= TRUE

<usr function> ::= USR<factor>

<val functi on> : := VAL<string-factor>

<vpos function> ::= VPOS


<chr string-func> ::= CHR$<factor>

<eval string-func> ::= EVAL<string-factor>

<fn string-func> ::= FN<variable name> [(<variable>
       {,<variable>})]

<get string-func> ::= GET$

<inkey string-func> ::= INKEY$<factor>

<left string-func> ::= LEFT$(<string>, <numeric>)

<mid string-func> ::= MID$(<string>, <numeric> [,<numeric>])

<right string-func> ::= RIGHT$(<string>, <numeric>)

<str string-func> ::= STR$[~]<factor>

<string string-func> ::= STRING$(<numeric>, <string>)
```