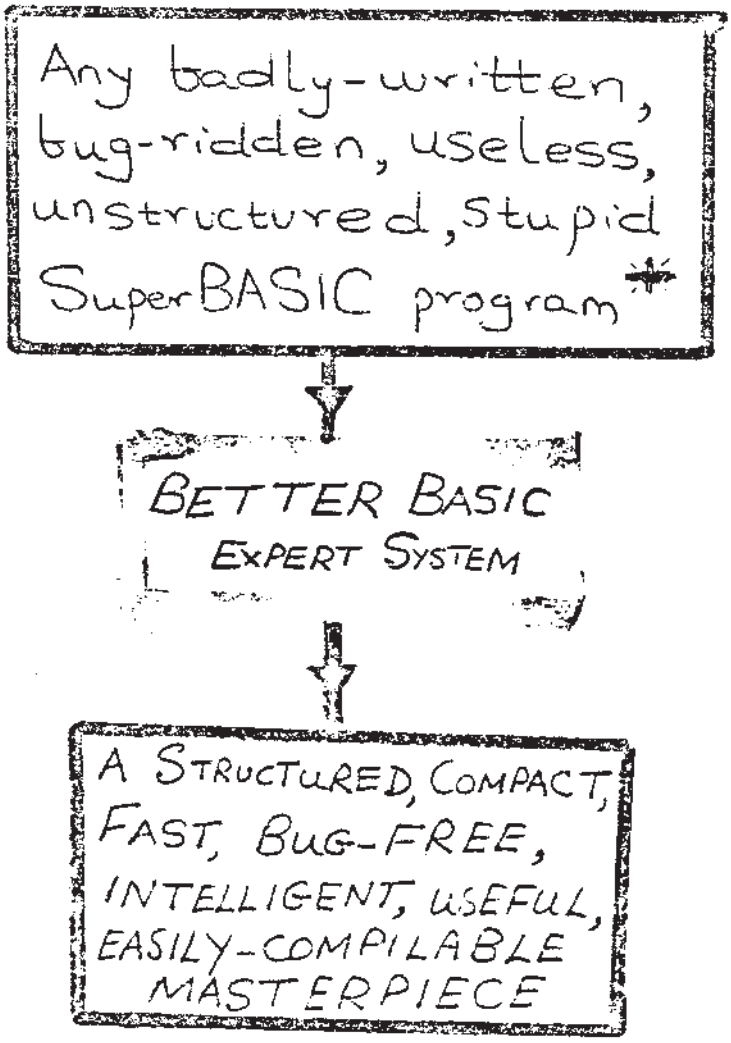
BETTER BASIC EXPERT SYSTEM

by

Charles Dillon

Published by Digital Precision, 222 The Avenue, London E4 9BE



The above is (well) almost true.

*even including programs by Ron Massey.....

1. What does BetterBasic do
2. Getting started
3. Program dialogue
4. Program operation and screen layout
5. Concepts and terms
6. Changes made by BetterBasic
7. Source code added to the output

1. What does BetterBasic do

BetterBasic reads the source code of a SuperBASIC program and examines the code for structural and other faults.

It creates a new version of the source, having tidied up the program by splitting long lines, indenting complex structures (REPeat loops, IFs, SELects etc). In addition any errors that may have been detected are either corrected or annotated.

The result is a program which is much easier to read, understand and maintain. In addition, the new source code is much less likely to give rise to compiler errors when SuperCHARGEd.

2. Getting started

BetterBasic is written in SuperBASIC and compiled with SuperCHARGE from Digital Precision.

As such the program is capable of being multi-tasked (i.e. does not have to be the only job running in the machine). It will be started with the standard system command EXEC or EXEC_W. BetterBasic uses some extensions to standard SuperBASIC, so if these are not in the machine, the extensions file on the supplied tape should first be loaded.

This start-up operation is accomplished using the supplied 'boot' program. The 'boot' procedure should be used if the machine has just been switched on, or if the system has been reset since the last time that BetterBasic was run. Under any other circumstance, it is sufficient to type "EXEC DVC_BetterBasic_bin", where DVC is the name of the device containing BetterBasic - e.g. mdv1 or flp2.

When the program has started, there is a short dialogue to establish the program to be processed, and the manner in which the user wants the program to be adjusted. After that, BetterBasic just gets on with it.

3. Program dialogue

3.1 "Enter input device type (f/m/k/r/w):"

A single key depression is required, and should be one of the letters indicated. If the key pressed is not one of these, then the first letter in the list ('f') is assumed. The significance of the letters is as follows:

f - flp
m - mdv
k - fdk
r - ram
w - win

3.2 "Enter input device number (1 - 6):"

A single key depression is required, and should be a number between 1 and 6. If the key pressed is none of these, then the value '2' is assumed.

3.3 "Enter input file name (_bas assumed):"

The file name of the program to be processed should be entered. BetterBasic expects that the name will be suffixed by _bas, and this suffix need not be typed.

The indicated file is opened, using the combined responses to the previous three prompts. If the device or file can not be accessed, the program reports an error ("File not found") and returns to the prompt at 3.1.

Having opened the input file, BetterBasic is able to establish how much workspace in memory will be needed to process the file. If the required memory is not free, the program terminates with the message "Insufficient memory available".

3.4 Output file identification

The same sequence of prompts is issued for the output file details. The file name prompt will allow a null response, in which case the same "root" is used for the output file name as was given as the input file name. The output file suffix is either "_rnm" or "_rfm" (see below).

The user should ensure that there is sufficient space on the output device to accommodate the program, allowing for the likely increase in program size (about 20%), and also the fact that some work space may be needed if the input program includes line number references ("GO TO" type statements). If the output device does not have much space, or if it is 'read only', the program will report "Make space on device" or "Can't open file". If the file already exists on the output device, it will be overwritten.

3.5 "Do you want input list (y/n):"

If the response is other than "n", the input source code will be displayed as it is read during subsequent processing.

3.6 "Do you want output list (y/n):"

If the response is other than "n", the output source code will be displayed as it is read during subsequent processing.

3.7 "Do you want to resequence (y/n):"

If the response is other than "n", the output source code will be resequenced. In that case, the user is asked to respond to the two subordinate prompts, as below.

3.7.1 "Enter start line no (default 100):"

Either a positive integer, or simply the ENTER key to accept the default, should be entered by the user.

3.7.2 "Enter line interval (default 10):"

Either a positive integer, or simply the ENTER key to accept the default, should be entered by the user.

BetterBasic checks that the combination of start line number and interval will not result in an invalid line number for any of the program lines in the output file. If check fails, an error message results, and the user is required to re-enter the last two responses.

3.8 **"Do you want to reformat (y/n):"**

If the response is other than "n", the output source code will be reformatted relative to the input source.

3.8.1 "Enter max line length (default 70):"

This prompt occurs if the explicit or implicit response to the reformat prompt was "y". The user is required to enter a positive integer less than 1000, or to accept the default by pressing ENTER.

3.9 **"Add warning text to program (y/n):"**

Where BetterBasic introduces new statements in the output source code, these statements may, at the user's choice, be accompanied by a comment. The comment will be of the form "rem *^* Added during reformat". Error messages - which have the form "mistake *^* some descriptive text" - included in the output will not be suppressed whatever the response to this prompt.

Program operation and screen layout

Having completed the initial dialogue, the main program activity commences.

The screen is cleared and a new heading displayed. This heading shows the size of the input file (in characters), the number of characters that have been processed, the number of warnings in the output source and the number of errors detected in the input source and commented in the output source.

Depending on the start-up options selected, the lower part of the screen may contain the following:

(a) Shown in white on red: A listing of each input line
(b) Shown in black on red: A listing of each output line

The individual listings may be toggled on/off as the program is running, using the keys TAB (input list) and ENTER (output list). If the screen gets corrupted (by another job in the machine), it may be redrawn by pressing the F4 key.

If any other key is pressed, the program will be suspended, with the message:

"Do you want to terminate now (n/y):"

If the response is other than "y", the program continues, otherwise the run is abandoned.

Depending on the nature of the input file and the start-up options selected, BetterBasic may need to make a second pass of the file. If so, the message "Second pass - renumber phase" is displayed and the output file is reprocessed by locating all "GO TO" type statements and adjusting the "old" line number reference to the correct value.

If this second pass is not required, the generated output will be in a file with the suffix "_rnm". If the second pass is required, the output file will have the suffix "_rfm".

The program ends with the message "End of reformat".

5. Concepts and terms

SuperBASIC is a very powerful and flexible programming language. It differs from standard BASIC at least to the extent that completely "structured" code may be written, without any requirement to make explicit line number references.

Conversely, it is possible to write programs in SuperBASIC that take advantage of none of these extra features offered by the language, and to continue using statements such as "GO TO", "GO SUB" and ignore the power of PROCEDURES and FUNCTIONS, and the associated ability to pass parameters, define LOCAL variables etc etc.

Because the SuperBASIC interpreter is trying to be all things to all men, and in a limited amount of ROM space in the QL, the interpreter allows as valid many constructs that would normally be regarded as invalid or illegal syntax.

This is a mixed blessing to the new or undisciplined programmer, since it permits the introduction of poorly written code which may contain "hidden" logic errors, resulting in mystifying behaviour of the program at run time.

BetterBasic is an attempt to "clean up" some of the most frequently occurring types of syntax and structural errors.

BetterBasic views a SuperBASIC program as a "structure".

A structure is an object that contains clauses. A clause is a collection of one or more program statements and/or structures.

The structures defined within the SuperBASIC language are introduced by one of the following statements:

IF, REPeat, FOR, SELect, DEFine, WHEN (not recommended)

Each structure which is introduced must also be terminated by the corresponding termination statement.

END IF, END REPeat, END FOR, END SELect, END DEFine, END WHEN

These terminators should be regarded as syntactic elements without any executive responsibility. It is simplest and safest to just believe that they have "got to be there", purely to indicate (to the interpreter or compiler and the programmer) where the structure ends. When it matters - which is not right now - a slightly different view may be taken.

General templates for the structures are:

```
IF condition
  true_clause : ELSE : false_clause : END IF

REPeat identifier
  repeated_clause : END REPeat identifier

FOR identifier = for_range_spec
  for_clause : END FOR identifier

SELect ON variable
  select_clause : END SELect

DEFine [ PROCedure | FuNction ] def_name
  definition_clause : END DEFine

WHEN condition
  when_clause : END WHEN
```

The most obvious rule from these templates is that the definition of each of the clauses is implied by the placement of the start of structure statement and the end of structure statement.

The second thing to notice is SuperBASIC's restriction that the "DEFine" structure statement may not appear in any structure other than the "top level" structure - namely the program itself. Any of the other structures may occur within any clause.

The third point is that for each start of structure statement there is only one end of structure statement possible - i.e. only one should physically occur in the program. Many SuperBASIC programmers seem to confuse the executive statements "NEXT" and "EXIT" with the syntax element "END". Clearly, any (subordinate) structure which has been properly defined will be completely contained within one clause (of another structure).

Most of these structures have a "short form", which tends to have been inherited or based on the corresponding form in standard BASIC.

In the short form, the start of structure statement and the following clause occur on the same line. In these cases, unfortunately, the end of structure statement is (syntactically) optional. The saddest part about the SuperBASIC interpreter is that it actually allows multiple short form structures on the same line, although this was not a design objective, merely a freak of coding.

Programs that utilise this freak are treading on thin ice, and are in any case not examples of good coding practice. BetterBasic modifies the output program to eliminate multiple short forms on one line.

These, then, are the structural principles that BetterBasic checks for, and where possible enforces. There are sundry other effects achieved by the program, most of which are designed to enforce or enhance the basic program structure. The detail of the changes made is described in the following section.

Changes made by BetterBasic

Statements per line

If the start-up option to reformat has been selected, BetterBasic will check that input lines do not violate the stated maximum line length. Lines found to be in excess of the maximum will be split to either minimise or eliminate the violation.

If the reformatting check results in extra lines being created in the output, BetterBasic will automatically "switch on" renumbering, irrespective of the start-up option selected.

If the line has to be split, BetterBasic also checks that where "short form" structures are being used, the line is split in the appropriate place, converting the "short form" to the "long form". The corresponding end of structure statement is automatically generated in the output source.

A further check for multiple short forms on the same line causes the line to be split even if it does not violate the maximum line length. In the output source, a start of structure statement will always occur as the first statement on a line.

2 **Indenting**

Each start of structure statement causes a two character indentation of the statements comprising the clause(s) of the structure.

The indentation is reduced by two characters after an end of structure statement has been written to the output.

This is to enhance the visual coherence of the associated statements within the structure.

For similar reasons of visual clarity, the executive statements within a SELect structure are indented beyond the SELect condition, assuming that the executive statements require more than one line.

Separation of statements " : "

To further aid the visual clarity of the output, successive statements on the same line are separated a colon between two spaces (" : ").

6.4 The noise word THEN

The noise word THEN is eliminated from the output source. This word is optionally used in "IF" statements. If a "short form" IF includes the word THEN, it is replaced by colon.

6.5 ELSE

The key word ELSE - if occurring as part of a statement - is generally replaced with the statement :ELSE. This in effect standardises the use of the ELSE key word, such that if other statements follow on the same line, the ELSE is always followed by a colon. (In many programs, the key word is sometimes used with a trailing colon and other times not).

6.6 Null statements

Spare colons are removed from the output (computerised surgery - whatever next). This covers situations where a line ends in one or more colons, and where successive statements on the same line are separated by more than one colon.

Where a colon occurs as the sole item on a line (effectively a comment line), the line is preserved.

6.7 NEXT => endfor

When the end of structure terminator for a "FOR" structure is written as "NEXT" (which is allowed but syntactically incorrect and is the cause of a lot of confusion), the output is amended to read "endfor".

6.8 Additional source code - Error checks

Various structure checks are made on the input source. If errors are detected, appropriate messages are included in the output source. The detail of the checks and the error messages is in the following section.

Source code added to the output

The following text will be added to the output source program, under the conditions indicated. A few of the tests are not relevant to a program properly developed using the SuperBASIC interpreter, since the syntax checks in the interpreter will trap certain types of mistake. However, BetterBasic is designed to process an input source file whatever its origin. It could for example have been generated in a text editor, without line numbers, and the keywords may be all in lower case or using the acceptable (to the interpreter) keyword abbreviations, such as

rep	short form for	REPeat
sel		SELect
endif, endfor, etc		END IF, END FOR etc
deffn, defproc		DEFine FuNction/PROCedure
goto, gosub		GO TO, GO SUB

Each of these short forms is expanded when the program is loaded into the interpreter, as may be seen via the LIST command.

Generally, BetterBasic will take SuperBASIC statements in a "raw" form and generate code acceptable to the interpreter.

In the following sections, messages which start with "rem *^*" are treated as warnings, and for each instance, the "warning" count at the top of the run-time screen is incremented.

Equally, the "errors" count on the run_time screen is incremented for each instance of a message starting with "mistake *^*".

The "*^*" is to enable the user to have a characteristic 'search pattern' to scan the output file - using an editor or similar.

7.1 "rem *^* Added during reformat"

If the comment option was selected at start-up, this comment will be appended to an end of structure statement which has been introduced by BetterBasic.

End of structure statements are added to the output source if a structure in the input has been split from a "short form" (single line) structure to a multi-line structure.

This will happen if either (a) the input source line contains multiple statements and is longer than the indicated maximum line length or (b) the input source line contains more than one structure. Whether comments have been requested or not, the end of structure statement will occur in the output.

The statements added are of the form:

```
endif
endsel
endfor nnnn     - where 'nnnn' is the structure identifier
endrep nnnn
endwhen
enddef
```

Example:

```
Input:  REPeat drain:IF NOT LEN(INKEY$(%1)):EXIT drain

Output: REPeat drain
          IF NOT LEN(INKEY$(%1)) : EXIT drain
          endrep drain : rem *^* Added during reformat
```

7.1.2 The same comment will occur as a separate line, assuming again that the comment option has been selected at start-up, if the input source contains a statement including a 'literal slice'. This feature is added because SuperCHARGE does not support sliced literals. BetterBasic does not presently support lines containing multiple literal slices.

Example:

```
Input:  day = (day-1) * 3 + 1
        PRINT "SunMonTueWedThuFriSat"(day to day+2)

Output: day = (day-1) * 3 + 1
        RFMT_temp$ = "SunMonTueWedThuFriSat"
        rem *^* Added during reformat
        PRINT RFMT_temp$(day to day+2)
```

7.2 "mistake *^* Following statement too long"

BetterBasic has an internal maximum of 300 characters for a single statement. If the input statement is in excess of this figure, then the statement is truncated at the maximum. The portion chopped off is displayed after the "mistake" line as a "rem" line, so that no source text is lost from the output file.

7.3 "mistake *^* Word in following statement too long"

In the present release of BetterBasic this message will not appear. It is essentially the same test as in 7.2, relative to another internal limit. At present, the maximum statement size and the maximum word size in BetterBasic are set to the same value (300), so 7.2 will occur.

7.4 "mistake *^* DUPLICATE 'ELSE' STATEMENT"

This message occurs if the input source contains an "IF" structure within which there is more than one "ELSE" statement. The "ELSE" statements after the first will be flagged with this message.

Example:

```
    Input:  IF condition
              perform_proc_A:ELSE
              ELSE:perform_proc_B:END IF

   Output:  IF condition
              perform_proc_A : else
              else : mistake *^* DUPLICATE 'ELSE' STATEMENT
              perform_proc_B : END IF
```

7.5 "mistake *^* 'ELSE' STATEMENT NOT EXPECTED"

The message will be included in the output where an "ELSE" statement is encountered in the absence of a current "IF" structure.

Example:

```
    Input:  fi answer > 0 : PRINT "Positive":ELSE PRINT "Zero/Neg"

   Output:  fi answer > 0 : PRINT "Positive" : else : mistake *^*
                                                      'ELSE' STATEMENT NOT EXPECTED
            PRINT "Zero/Neg"
```

7.6 "mistake *^* STATEMENTS AFTER 'GOTO' CAN NOT BE REACHED"

If multiple statements are used on a program line, and a "GO TO", "NEXT" or "RETurn" statement is encountered as other than the last executive statement in a clause, the message will be included in the output.

Example:

```
    Input:  IF n > max
            GO TO 1210:x=x+4:ELSE NEXT r_loop:END IF

   Output:  IF n > max
              GO TO 1210 : mistake *^* STATEMENTS AFTER 'GOTO' CAN NOT
                                                                BE REACHED
              x=x+4 : else : NEXT r_loop : END IF
```

7.7 "mistake *^* INCORRECT SYNTAX"

The word "END" has been encountered in the input source, without a recognisable qualifier.

Example:

```
    Input:  IF NOT "key" INSTR inventory$
            PRINT "You can't open the box" : end f
            RETurn

   Output:  IF NOT "key" INSTR inventory$
              PRINT "You can't open the box" : end f
              mistake *^* INCORRECT SYNTAX
              RETurn
```

7.5 "mistake *^* 'ELSE' STATEMENT NOT EXPECTED"

The message will be included in the output where an "ELSE" statement is encountered in the absence of a current "IF" structure.

Example:

```
    Input:  fi answer > 0 : PRINT "Positive":ELSE PRINT "Zero/Neg"

   Output:  fi answer > 0 : PRINT "Positive" : else : mistake *^*
                                                    'ELSE' STATEMENT NOT EXPECTED
            PRINT "Zero/Neg"
```

7.6 "mistake *^* STATEMENTS AFTER 'GOTO' CAN NOT BE REACHED"

If multiple statements are used on a program line, and a "GO TO", "NEXT" or "RETurn" statement is encountered as other than the last executive statement in a clause, the message will be included in the output.

Example:

```
    Input:  IF n > max
            GO TO 1210:x=x+4:ELSE NEXT r_loop:END IF

   Output:  IF n > max
              GO TO 1210 : mistake *^* STATEMENTS AFTER 'GOTO' CAN NOT
                                                          BE REACHED
              x=x+4 : else : NEXT r_loop : END IF
```

7.7 "mistake *^* INCORRECT SYNTAX"

The word "END" has been encountered in the input source, without a recognisable qualifier.

Example:

```
    Input:  IF NOT "key" INSTR inventory$
            PRINT "You can't open the box" : end f
            RETurn

   Output:  IF NOT "key" INSTR inventory$
              PRINT "You can't open the box" : end f
              mistake *^* INCORRECT SYNTAX
              RETurn
```

"mistake *^* UNMATCHED STRUCTURE IDENTIFIERS"

This message occurs in the output when an "END FOR" or "END REPeat" statement is encountered with a structure identifier different from the current structure. The 'incorrect' statement will be displayed prior to this message. After the message, another comment line is displayed, showing the expected identifier. The alteration of indent assumes that the current structure has been ended.

Example:

```
Input:  i=0:REPeat num_scan
        i=i+1:IF i > LEN(str$):EXIT num_scan
        IF NOT str$(i) INSTR "0123456789":EXIT num_scan
        END REPeat num_sacn

Output: i=0
        REPeat num_scan
          i=i+1
          IF i > LEN(str$) : EXIT num_scan
          IF NOT str$(i) INSTR "0123456789" : EXIT num_scan
          END REPeat num_sacn
        mistake *^* UNMATCHED STRUCTURE IDENTIFIERS
        rem *^* Expected: 'endrep num_scan'
```

"mistake *^* UNMATCHED STRUCTURE TYPES"

A legal end of structure statement has been read, but it does not correspond to the current structure. The subsequent indentation implies that the current structure has not been ended.

Example:

```
Input:  FOR i=1 TO 10:
        IF i>4:PRINT TO 20;a$(i):ELSE:PRINT a$(i):END IF:END REPeat i
        j=12

Output: FOR i=1 TO 10
          IF i>4 : PRINT TO 20;a$(i) : else : PRINT a$(i) : END IF
          END REPeat i : mistake *^* UNMATCHED STRUCTURE TYPES
          rem *^* Expected: 'endfor i'
          j=12
```

7.10 **"mistake *^* CHANGED 'END' TO 'NEXT'; MAY NEED 'EXIT' INSTEAD**

This message occurs if an end of structure statement is detected within a conditional clause. This construction is (should be) syntactically invalid, but SuperBASIC allows it to be typed ... it will not necessarily process it correctly.

What the programmer must mean is either one of "NEXT" - most probable - or "EXIT" - also possible.

Example:

```
    Input:  FOR n=1 TO 256
            do_something
            IF b <= 0 : END FOR n
            do_something_else
            NEXT n

   Output:  FOR n=1 TO 256
              do_something
              IF b <= 0 : next n : mistake *^* CHANGED 'END' TO 'NEXT;
                                                  MAY NEED 'EXIT' INSTEAD
              do_something_else
              endfor n
```

7.11 **"mistake *^* Line number not found: xxxx,xxxx,..."**

If the input program contains references to line numbers (God forbid), then the generated source code is re-examined if and only if the line structure of the output differs from the input - which is almost certain to be the case, due to line-splitting, lines added by BetterBasic etc.

The message occurs if the input source has a line number reference which refers to a line not found in the input program. In SuperBASIC, such references will find their destination at the 'next higher' line than the one referred to. BetterBasic modifies the source such that the destination line is explicit.

Example:

```
    Input:  10 REMark Start of program
            .....
            .....
            124 GO TO 1

   Output:  100 REMark Start of program
            .....
            .....
            2010 GO TO 100 : mistake *^* Line number not found: 1
```

2 "rem *^* Not SuperCHARGEable"

These messages appear only if the comment start-up option was selected. They are used to highlight program statements that are not supported by the SuperCHARGE compiler. These are typically statements that refer to the modification or representation of program source lines - which is of course not meaningful after the program has been compiled.

Commands such as RENUM, DLINE, AUTO, EDIT, LIST etc will be highlighted.

Equally, in the present release of SuperCHARGE, the commands LRUN and MRUN are not supported - a similar effect being achieved in other ways.

If these statements are left in a program which is to be SuperCHARGEd, the compiler will terminate, having reported

"Command meaningless if compiled"