Feature

PreBASIC:

A Preprocessor For BASIC In BASIC

by Joseph A. Sabin, Jr.

PreBASIC is a preprocessor for BASIC written in BASIC. Input files for this preprocessor are ASCII text files with no line numbers. Multiple program files can be appended to each other with PreBASIC. These can be of modular nature, with labelled routines to be called by other portions of the final program. There is no limit to the number of files that can be joined, with a stack limit of about 10 levels of files.

PreBasic uses the following file types:

- .BBS source file.
- .INT intermediate 1st pass file.
- .BAS standard BASIC ASCII program file.

The concept is derived from a number of BASIC preprocessors I have seen or read reviews of. It is in the spirit of public domain software that this BASIC preprocessor is being published. I feel this preprocessor allows a natural program creation to occur.

The added syntax is very much like Pascal; the use of square brackets and parentheses are reversed. This reversal fits the nature of BASIC's array structure, so procedures and routines are called with square brackets defining the parameters to be passed to them. The main difference is that a procedure need not be defined in the code before being used. PreBASIC globally reviews all labels, thus allowing global procedure calls.

The program

The listing uses a number of comments to divide up the logical segments of the program. Most of the segments are comment free, but the code should be understandable when viewed with this text.

In the beginning of the listing (lines 1000-1050) we have the start-up code. Following this is a section that will read in the file name and starting line number and increment (1060,1130-1170), if you choose to use the command line option. This is only useful if you have a compiled version of PreBASIC.

CP/M COMMAND LINE SYNTAX

```
A>PB FILENAME 1000 10
```

If PreBASIC is not compiled, or you choose to start it without an extended command line, then the next section of code (1070-1110) requests the information needed. Line Inputs are used for neatness and therefore string values are changed to numerics.

First, PreBASIC makes sure the file name is all upper case. This upper case subroutine (2030-2080) is used often in PreBASIC. If there is no period in the file name, PreBASIC appends .BBS* to the file name. The main processing of the file then begins (1250).

PreBASIC loads in a file 100 lines at a time (1300-1430). If the file ends before 100 lines, PreBASIC closes that file and goes back to the previous file if there was one, or else ends this phase of processing. It is also during this phase that PreBASIC reviews the dot commands (1340-1380). Once the buffer of 100 lines is full or EOF (end of file), then the lines are parsed to upper case (2100-2170) and all labels are recorded. The line numbers are appended and each line is stored in the intermediate file *.INT. After all 100 lines are processed the file read process begins again.

PreBASIC now starts on the intermediate file (1540) to exchange all the requests for labels (1640-1810) with the proper line numbers. This also is done 100 lines at a time (1590-1630). All labels are replaced with the correct line number, and if there are parameters they are set into the line as equates to one another.

The file is stored to *.BAS as a standard MBASIC ASCII program file, 100 lines at a time (1770). This program can now be run with the interpreter or compiled. If the compressed file format is desired, it will be necessary to load the file into MBASIC and save it again in the compressed format.

When processing is complete you will be informed that the file has been completed with a message listing the number of errors reported and the output file name:

```
0000 ERROR(S)
Output File *.BAS
```

The number of errors will depend on how many errors were found in the file as it was processed. All the errors will have a commented line following them telling you what went wrong. Sometimes this error condition is easily fixed in the *.BAS file. A missing label can often be fixed in the *.BAS file, but the best place to do the repairs is in the source file *.BBS since then you will always have a complete source.

ROUTINES USED DURING PROCESSING

Parse Label

This subroutine (1830-1950) is looking for the end of the label following a dot. The various pointer variables (PNTR) are set to indicate the positions of the important pieces of information that might be contained here. These might be a series of parameters, and of course the full name of the label being called.

Find Label

Once the routine above finds the full name of the label, this subroutine (3240-3290) is called. This is a very straightforward binary search of the sorted array LBL$( ) for the label in question. The variables TOP, BOT, LOOK, and FLAG are used to indicate the obvious portions of any binary search routine. Once found (or not) FLAG indicates the number or the label in the array LBL$( ). If FLAG is set to zero then the label was not found.

Sort Labels

Before the labels can be searched they must be in order. I chose ascending order, and this routine does that (2320-2390). A Shell-Metzner sort is used for its virtues of speed and compactness.

Label Routine

When a square bracket is found as the first character in a line this routine is called (2190-2300). The line is then parsed to find the ending bracket and any parameters that are to be ►

*.BBS is the default file name convention for PreBASIC input files.

transferred. The array LIN( ) is used for the line number, LBL$( ) for the label's name, and LVR$( ) for the label variables if any.

Label with Parameters

This is the most complex portion of the listing (2410-2670). It is here that a series of parameters in a label get parsed. What I do is set a series of pointers to all the important sections of the parameter list (e.g., :'s and the / character). As these pointers are created and the variables are therefore separated from one another, the next subroutine is called.

Label Value Exchange

Set all variables in the called list equal to the calling list one at a time (2690-2810). This sets up the equates in the current line. When this subroutine is completed a line looks something like the example of parameter exchange later in this article. There is a limit of 10 variables in and 10 variables out. This limit is imposed by the arrays PNTR1( ) and PNTR2( ). If you exceed this limit, PreBASIC halts with an error. It is left to the programmer to avoid exceeding this limit.

Hex File Conversion

Small machine language routines are often most easily added into BASIC code as a series of DATAs, READs and POKEs. This subroutine (2830-3220) reads in a hex file whose name is presented with the dot command H. This routine very simply reads in the file as ASCII and translates the HEX ASCII two-byte strings to their integer equivalent. This is done by appending &H to the beginning of the HEX string and getting the VAL( ) of the string. The values are then transferred into a series of DATA statements of sixteen values or less.

The routine is made a bit more complex because of the possibility of multiple ORG statements in the assembly language routine, requiring the code to be placed in more than one memory location. It is up to the programmer to create a CLEAR statement to protect the machine code.

New File Name

This is a simple routine (3310-3360) to set up the new file name when the dot command I is encountered. The routine first determines the last character in the new file name (3320). If you want to comment this line you must leave a space after the file name. The file name is then set to upper case (gosub 2040), and then parsed for a dot. If no dot is found then the default file type .BBS is added. This new file is now the current working file until its EOF is reached.

Print Error Message

Error messages are printed to the console and placed into the .BAS code if the line increment is greater than one (3380-3450). The error message is given a line number one greater than the line with the error in it. The line is a comment and a caret (↑) points to the place the error occurred. Variables ERT (error tab), ER (error number), and FLAG are set before calling this routine. ERT equals the number of characters from the left hand side of the current line to the error position. ER is the error number that matches the error to the error statement in the DATA line (1030).

Change Line Number

The dot command "R" calls this subroutine (3470-3530) to change the current line number LN and the line number increment IN. You have the option of changing the line number alone or both line number and line increment.

I hope the routines in this program have been made clear by my discussion of them. They have been handled in order of use, then in order of their appearance in the code if they are not used in any particular order. In some of the routines, error checking is done to prevent the program from saving a "mistake," rather than expanding code to make it impossible. This is most obvious in lines 2540 and 2570, when a double colon or colon equal sign can be placed together. This is a lack in the parser, but it makes the code smaller and a bit faster by not having to be too careful. The final results are accurate.

PREBASIC SYNTAX

Dot commands are used to invoke different functions of PreBASIC. A space is required after the dot command before the parameters of the command are given. As with WordStar, the dot must be the first character on a line.

.H Hex file for FOR . . . POKE . . . NEXT structure from a standard Intel Hex file.
.I Input of Include file name.
.R Renumber line number and increment.

Dot Command Examples

Hex file load at this position in code

```
.H SORT.HEX
```

A file created by ASM will be of the correct HEX format. This is the only type of file I have tested this routine on. I assume it will work on any HEX type file which follows the same format.

When a hex file is loaded you will have code that looks something like this:

```
1130 DATA 123,45,67,89,121,33,34,45,56,67,78,1,2,3,4,5
1140 DATA 55,66,77,88,99
1150 FOR HEX = 1024 TO 1044
1160 READ HEX1
1170 POKE HEX,HEX1
1180 NEXT
```

If the code skips a memory location, then a second or third, etc. set of FOR . . . POKE . . . NEXT will be created in the listing. This type of code will occur whenever you use two ORG statements, or for some other reason the HEX file doesn't finish a full line of 16 values before beginning another.

Include a file in the code at this position

```
.I INCLUDE.BAS
```

This will cause PreBASIC to leave the current file and open the file addressed in the include statement. The "Include" file will be loaded and sequentially numbered until the "End of file" is reached. Upon end of file, the previous file will be input from the line after the include line.

PreBASIC allows the nesting of Include files to the file buffer limit you started with in MBASIC (/f:15 is the upper limit). The include function is implemented to make the structure of the code easier. Routines used many times can be saved as a file and used without regard to the variables needed (of course they cannot conflict with other variables in the program).

Renumber Current Line and Increment

```
.R 10000 100
```

The ability to change the numbers in a program to suit the needs of a structure is done with Renumber. When the ".R" command line is found, PreBASIC reads in the two values and changes the current line variable to the first value and the current line increment variable to the second value. If the current line number is greater than the requested line number you will be informed of a line number crash and your request will be ignored. If you do not include a second number the current line increment value will not be changed. The syntax requires you to use a space to delimit the two numbers.

Renumber can also be used in the same manner as the command line option. If you want the simplest manner of running PreBASIC the use of "R" as one of the first lines in the code will turn on the line numbering you want instead of the default first line number of 10 and default line increment of 10.

Commenting

```
; PreBASIC Only comment
```

A PreBASIC Only source comment allows you to use inline documentation without cluttering up the final code. These lines are just ignored by the preprocessor. The choice of a semicolon follows the use in the CP/M assembler. This type of comment can be used as a sort of pseudo-code before you write the actual code. For example:

```
; open the program with a signon
;    get user input what to do
;      go do what the user wants
;    continue
; end
```

From this point you would then fill in the code for PreBASIC to do what it is you wanted, for example:

```
; open the program with a signon
[TOP]PRINT "Hello there Folks"
;    get user input what to do
     INPUT "What would you like to do";A$
;       go do what the user wants
        GOSUB .THING_TO_DO
;    continue
     GOTO .TOP
[THING_TO_DO]FOR I = 1 TO 10
     PRINT "Testing . . . Line #";I
NEXT
RETURN
; end
END
```

This would then be given to PreBASIC and the following code would result:

```
10   PRINT "Hello there Folks"
20      INPUT "What would you like to do";A$
30         GOSUB 50
40      GOTO 10
50   FOR I = 1 TO 10
60      PRINT "Testing . . . Line #";I
70   NEXT
80   RETURN
```

If standard remarks were in place they would also appear in the final code. If the code needs to send a value from a variable to the corresponding one in the routine it is calling, all you need to do is list the variables. A label to be called is to have the following syntax.

Labels

```
[Label_One]
[Label_Two PARAM1:PARAM2/]
[Label_Three PARAM/RESULT]
```

A label's bracket must be the first thing on the line. The line can be tabbed in from the closing of the label (the right bracket). The parameters to be passed to and from labelled routines must match in quantity. PreBASIC does not check to see if they match in type; that is up to the programmer (because the programmer can define a type for a variable group). A label can be called from within a program in the following situations:

```
  ELSE .LABEL_NAME
GOSUB .LABEL_NAME
  GOTO .LABEL_NAME[PARAMS/]
  THEN .LABEL_NAME
```

Square brackets are for parameters. The dot preceding the label name allows you to use standard line numbers if you want. The line is parsed looking for the above four "label callers," a space then a dot. If those requirements are met, the characters following until a space, carriage return or opening square bracket are included in the label. A label must not have a space; any other character will do. Case is ignored (e.g., top and TOP are equal).

The syntax for exchange is as follows:

```
GOSUB .SORT[GO1:GO2/GET1]
.
.
.
[SORT GET2:GET3/GO3]'again a comment
.
.
.
RETURN
```

This will process down to:

```
1000   GET2 = GO1:GET3 = GO2:GOSUB 1500:GET1 = GO3
.
.
.
1500   'again a comment
.
.
.
1600   RETURN
```

PreBASIC automatically exchanges the variables listed in the variable area of the label. If there is a parameter mismatch PreBASIC will inform you of the error and save the error message in the output file.

The delimiter is a colon ":" between variables, and a slash "/" between output values and input values. The slash is a required element in the exchange of values even if there are only input or output values.

There can be exchange of values into and out of file routines. This is done in the same manner as for any label exchange of values. The creation of the code is done after all the lines of code have been merged into a single intermediate source file. Once this file has been created, all labels are translated into line numbers and variable exchange code.

Programming with PreBASIC requires a word or text processor of some sort. When you are writing code for the new program you will be able to add lines at will anywhere in the program without regard to line numbers. The creation of a GOTO or GOSUB is simply the addition of a dot, then the name of the routine to be jumped to.

Often-used routines should be written using unique variable names, so as to avoid conflict in variables when they are used in a program written afterwards. A method I have used is to put the routine's name first in all variables, then use a simple letter number variable appended to the end, e.g., routine SORT would have variables named SORTA, SORTB, SORTB1, etc. This will usually avoid the repetition of variable names in a program using included routines.

Obviously there can be multiple extensions to this program, some of which I have already thought of. The easiest way to extend this program is to use it to preprocess itself after the extensions have been added. Currently I am working on a vastly expanded version which will be offered as a compiled program with PreBASIC source. This way you will be able to add to your heart's content. When this is available, I will announce it in *Lifeline/The Software Magazine.* Method of distribution has not been decided, and the product is not yet available. Version 1.04 (the one that follows) ought to keep you busy for a while. ►

Listing 2 is a small test program showing off some of the more interesting features of PreBASIC. Give it a try. It is also a good test to see if PreBASIC is really working right. Listing 3 is the resulting PreBASIC output.

Listing #1 PreBASIC Version 1.04

```
1000  '---------- OPEN PROGRAM SIGNON ETC. ----------
1010  DEFINT A-Z
1020  DIM LP$(105),LIN(100),LBL$(100),LVR$(100),F$(15),PNTR1(10),
      PNTR2(10),L1$(20),L2$(20),ER$(10)
1030  DATA "Label not found" , "odd number of quotes" , "label did not
      end" , "Parameter list does not match" , "out of Library stack
      space" , "library routine not found" , "no input file name" , "line
      number crash request ignored" , "file is nonexistent"
1040  FOR I = 1 TO 9:READ ER$(I):NEXT
1050  PRINT "(c) 1984 MicroFrame Associates - Version 1.04":PRINT:LP = 1
1059  '---              ↓ - CP/M'S PLACE FOR COMMAND LINE INFO ---
1060  IF PEEK(&H80)<>0 THEN GOTO 1130
1069  '---------- HAND INPUT OF PARAMETERS ----------
1070  LINE INPUT "File name to READ = ";F$
1080  LINE INPUT "Beginning Line # = ";LI$:LI = VAL(LI$)
1090  LINE INPUT "Line # Increment = ";IN$:IN = VAL(IN$)
1100  IF LI = 0 THEN LI = 10
1110  IF IN = 0 THEN IN = 10
1120  GOTO 1180
1130  FOR I = 1 TO PEEK(&H80):TEMP$ = TEMP$ +
      CHR$(PEEK(&H80 + I)):NEXT
1140  IF LEFT$(TEMP$,1) = " " THEN TEMP$ = MID$(TEMP$,2)
1150  PNTR = INSTR(TEMP$," "):IF PNTR = 0 THEN
      F$ = TEMP$:GOTO 1180 : ELSE
      F$ = LEFT$(TEMP$,PNTR-1):TEMP$ = MID$(TEMP$,PNTR + 1)
1160  PNTR = INSTR(TEMP$," "):IF PNTR = 0 THEN LI = VAL(TEMP$)
      ELSE LI = VAL(LEFT$(TEMP$,PNTR))
1170  IF PNTR<>0 THEN IN = VAL(MID$(TEMP$,PNTR + 1))
1179  '---------- MAKE THE FILE NAME BE UPPER CASE ----------
1180  GOSUB 2040
1189                  ↓ SEE IF THERE IS A DOT - ADD .BBS IF NOT
1190  P = INSTR(F$, " . "):IF P = 0 THEN F$ = F$ + '.BBS'
1199  '------              ↓ ------ AND DO IT AGAIN AND ADD .INT ----
1200  P = INSTR(F$," . "):FO$ = LEFT$ (F$,P) + "INT"
1210  IF LI = 0 THEN LI = 10
1220  IF IN = 0 THEN IN = 10
1230  LN = LI-IN:FIL = 2:F$(FIL) = F$
1240  '---------- MASTER FILE READER ----------
1250  PRINT "Processing File ";F$(FIL);CHR$(13);
1260  OPEN "O",1,FO$
1270  OPEN "R",FIL,F$(FIL):IF LOF(FIL) = 0 THEN
      ERT = 0:ER = 9:GOSUB 3380:
      LP$(LP) = AB$:LP = LP + 1:LP$(LP) = AC$:GOTO 1480
1280  CLOSE #FIL
1290  OPEN "I",FIL,F$(FIL)
1300  WHILE (EOF(FIL)<> - 1 AND LP< = 100)
1310    LN = LN + IN
1320    LINE INPUT #FIL,AA$
1330    GOSUB 2100
1340    IF LEFT$(AA$,2) = ".I" THEN F$(FIL + 1) = MID$(AA$,4):FIL =
        FIL + 1:GOSUB 3310 :PRINT "Processing File ";
        F$(FIL);SPC(12-LEN(F$(FIL)));CHR$(13);:LN = LN-IN:GOTO 1270
1350    IF LEFT$(AA$,2) = ".H" THEN FIL = FIL + 1:GOSUB 2830 :GOTO
        1430 :'HEX FILE
1360    IF LEFT$(AA$,1) = "[" THEN GOSUB 2190 :GOTO 1400
1370    IF LEFT$(AA$,1) = ";" THEN LN = LN-IN:GOTO 1300
1380    IF LEFT$(AA$,2) = ".R" THEN GOSUB 3470 :GOTO 1300
1390    A$ = MID$(STR$(LN),2) + " " + AA$
1400    IF A$ = MID$(STR$(LN) + " " ,2) THEN A$ = A$ + "empty line —
        check goto's etc. before deleting"
1410    LP$(LP) = A$
1420    LP = LP + 1
1430  WEND
1440  IF EOF(FIL) = - 1 THEN 1480
1450  FOR I = 1 TO LP:PRINT #1,LP$(I):NEXT
1460  LP = 1
1470  GOTO 1300
1480  CLOSE #FIL
1490  IF FIL>2 THEN FIL = FIL - 1:GOTO 1300
1500  FOR I = 1 TO LP:PRINT #1,LP$(I):NEXT
1510  IF FIL>2 THEN FIL = FIL - 1:GOTO 1300
1520  F$ = FO$:CLOSE:GOSUB 2320
1530  '---------- PROCESS LABELS ----------
1540  FOUT$ = LEFT$(FO$,LEN(FO$) - 3) + "BAS"
1550  PRINT SPC(30);CHR$(13);
1560  OPEN "I",2,F$
1570  OPEN "O",1,FOUT$
1580  LP = 1
1589  '---------- LOAD IN 100 LINES ----------
1590  WHILE (EOF(2)<> - 1 AND LP< = 100)
1600    LINE INPUT #2,AA$
1610    LP$(LP) = AA$
1620    LP = LP + 1
1630  WEND
1639  '---------- NOW PROCESS 100 LINES ----------
1640  LP = LP - 1
1650  FOR INCR = 1 TO LP
1660    PRINT CHR$(13);"Processing line #";(VAL(AA$));
1670    AA$ = LP$(INCR)
1680    A$ = AA$
1690    PNTR = INSTR(A$,"THEN . ")
1700    IF PNTR<>0 THEN 1840
1710    PNTR = INSTR(A$,"ELSE . ")
1720    IF PNTR<>0 THEN 1840
1730    PNTR = INSTR(A$,"GOTO . ")
1740    IF PNTR<>0 THEN 1840
1750    PNTR = INSTR(A$,"GOSUB . ")
1760    IF PNTR<>0 THEN 1830
1770    PRINT #1,AA$
1780  NEXT
1790  IF EOF(2)<> - 1 THEN LP = 1:GOTO 1590
1800  CLOSE
1810  GOTO 1970
1820  '---------- PARSE LABEL ----------
1830  PNTR = PNTR + 7:J = 7:GOTO 1850
1840  PNTR = PNTR + 6:J = 6
1850  PNTR1 = PNTR + 1:PNTR2 = 0
1860  IF MID$(A$,PNTR1,1) = " " THEN PNTR2 = PNTR1
1870  IF MID$(A$,PNTR1,1) = ":" THEN PNTR2 = PNTR1
1880  IF MID$(A$,PNTR1,1) = "[" THEN PNTR2 = PNTR1:GOTO 2410
1890  IF PNTR2 = 0 THEN PNTR1 = PNTR1 + 1:IF PNTR1>LEN(A$)
      THEN PNTR2 = PNTR1:GOTO 1900 ELSE 1860
1900  A$ = MID$(A$,PNTR,PNTR2-PNTR):FLAG = 0
1910  GOSUB 3240  :'FIND LABEL
1920  IF FLAG = 0 THEN ERT = PNTR:ER = 1:GOSUB 3380:
      PRINT #1,AB$:AA$ = AC$:GOTO 1770
1930  AA$ = MID$(AA$,1,PNTR-2) + MID$(STR$(LIN(FLAG)),2) + " " +
      MID$(AA$,PNTR2)
1940  IF FLAG>0 THEN 1950 ELSE 1680
1950  GOTO 1680
1960  '---------- END ----------
1970  ER$ = RIGHT$("0000" + MID$(STR$(NERROR),2),4)
1980  PRINT :PRINT ER$;" Error(s)"
1990  PRINT "Output File ";FOUT$
1999  '---------- DISPOSE OF THE .INT FILE ----------
2000  KILL FO$
2010  END
2020  '---------- UPPER ----------
2030  'THIS TURNS A LOWER CASE FILE NAME INTO AN UPPER
      CASE NAME
2040  FOR I = 1 TO LEN(F$)
2050    G$ = MID$(F$,I,1)
2060    IF G$> = "a" AND G$< = "z" THEN MID$(F$,I,1) =
        CHR$(ASC(G$)AND 95)
2070  NEXT
2080  RETURN
2090  '---------- PARSE LINE - DO UPPER CASE ----------
2100  P = INSTR(AA$,CHR$(34)):PB = 1:E$ = " "
```

```
2110  IF P = 0 THEN F$ = AA$:GOSUB 2040 :AA$ = F$:RETURN
2120  P1 = INSTR(P + 1,AA$,CHR$(34))
2130  IF P1 = 0 THEN ERT = 0:ER = 2:GOSUB 3380:
      LP$(LP) = AB$:LP = LP + 1:AA$ = AC$:RETURN
2140  F$ = MID$(AA$,PB,P-PB):GOSUB 2040:
      E$ = E$ + F$ + MID$(AA$,P,P1-P + 1):PB = P1 + 1
2150  P = INSTR(PB,AA$,CHR$(34))
2160  IF P = 0 THEN F$ = MID$(AA$,PB):GOSUB 2040:
      E$ = E$ + F$:AA$ = E$:RETURN
2170  GOTO 2120
2180  '---------- LABEL ROUTINE ----------
2190  P = INSTR(AA$,"]")
2200  PARM = INSTR(AA$,"  ")
2210  IF PARM<>0 AND PARM<P THEN PARM1 = INSTR(AA$,"]") ELSE
      LVR$(LIN + 1) = " ":PARM = 0:P1 = 0:GOTO 2240
2220  LVR$(LIN + 1) = MID$(AA$,PARM + 1,PARM1 - PARM - 1)
2230  P1 = PARM
2240  IF P = 0 THEN ERT = 0:ER = 3:GOSUB 3380 :RETURN
2250  IF P1 = 0 THEN P1 = P
2260  A$ = MID$(STR$(LN),2) + " " + MID$(AA$ + " " ,P + 1)
2270  IF RIGHT$(A$,2) = " " THEN A$ = LEFT$(A$,LEN(A$) - 1)
2280  LIN = LIN + 1
2290  LIN(LIN) = LN:LBL$(LIN) = MID$(AA$,2,P1 - 2)
2300  RETURN
2310  '---------- SORT LABELS ----------
2320  M = LIN
2330  M = INT(M/2):IF M = 0 THEN 2390 ELSE K = LIN-M:J = 1
2340  I = J
2350  L = I + M:IF LBL$(I)<=LBL$(L) THEN 2380
2360  SWAP LBL$(I),LBL$(L):SWAP LIN(I),LIN(L):SWAP LVR$(I),LVR$(L)
2370  I = I - M:IF I>=1 THEN 2350
2380  J = J + 1:IF J>K THEN 2330 ELSE 2340
2390  RETURN
2400  '---------- LABEL W/ PARAM(S) ----------
2410  A$ = MID$(A$,PNTR,PNTR2-PNTR):FLAG = 0
2420  AB$ = LEFT$(AA$,PNTR - J - 1)
2430  GOSUB 3240 :'FIND LABEL
2440  IF FLAG = 0 THEN ERT = PNTR:ER = 1:GOSUB 3380 :PRINT
      #1,AB$:AA$ = AC$:GOTO 1770
2450  PNTR4 = INSTR(PNTR2,AA$,"]") - 1
2460  GG$ = MID$(AA$,PNTR2 + 1,PNTR4-PNTR2)
2470  PNTR3 = INSTR(PNTR2,AA$,"]")
2480  PNT1 = INSTR(GG$,"/")
2490  GA$ = MID$(GG$,1,PNT1 - 1)
2500  PNT3 = INSTR(LVR$(FLAG),"/")
2510  GB$ = MID$(LVR$(FLAG),1,PNT3 - 1)
2520  IF LEN(GA$)<>0 THEN GOSUB 2690 :IF FAULT = 1 THEN
      FAULT = 0:GOTO 1770
2530  AB$ = AB$ + MID$(AA$,PNTR - J,J - 1) +
      MID$(STR$(LIN(FLAG)), 2) + ":"
2540  G = INSTR(AB$,"::")
2550  IF G<>0 THEN AB$ = LEFT$(AB$,G) + MID$(AB$,G + 2)
2560  IF G<>0 THEN 2540
2570  G = INSTR(AB$,": = ")
2580  IF G<>0 THEN AB$ = LEFT$(AB$,G - 1) + MID$(AB$,G + 1)
2590  IF G<>0 THEN 2570
2600  GB$ = MID$(GG$,PNT1 + 1)
2610  GA$ = MID$(LVR$(FLAG),PNT3 + 1)
2620  IF LEN(GA$)<>0 THEN GOSUB 2690
2630  IF RIGHT$(AB$,1) = ":" THEN AB$ = LEFT$(AB$,LEN(AB$) - 1)
2640  IF PNTR3 = LEN(AA$) THEN AA$ = AB$:GOTO 1770
2650  AA$ = AB$ + MID$(AA$,PNTR3 + 1)
2660  GG = PNTR3 + 1
2670  GOTO 1680
2680  '---------- LABEL VALUE EXCHANGE SECTION ----------
2690  A = 0:FOR I = 1 TO LEN(GA$)
2700    IF MID$(GA$,I,1) = ":" THEN A = A + 1:PNTR1(A) = I
2710  NEXT:IF A = 0 THEN PNTR1(1) = LEN(GA$) ELSE
      PNTR1(A + 1) = LEN(GA$)
2720  B = 0:FOR I = 1 TO LEN(GB$)
2730    IF MID$(GB$,I,1) = ":" THEN B = B + 1:PNTR2(B) = I
2740  NEXT:IF B = 0 THEN PNTR2(1) = LEN(GB$) ELSE
      PNTR2(B + 1) = LEN(GB$)
2750  IF A<>B THEN ERT = PNTR:ER = 4:GOSUB 3380 :PRINT
      #1,AB$:AA$ = AC$:FAULT = 1:RETURN
2760  C = 1:D = 1
2770  FOR I = 1 TO A + 1
2780    AB$ = AB$ + MID$(GB$,D,PNTR2(I) - D + 1) + " = " +
        MID$(GA$, C,PNTR1(I) - C + 1) + ":"
2790    C = PNTR1(I) + 1:D = PNTR2(I) + 1
2800  NEXT
2810  RETURN
2820  '---------- HEX FILE CONVERT ----------
2830  PK1 = 0:PK2 = 0
2840  F$(FIL) = MID$(AA$,4)
2850  OPEN "R",FIL,F$(FIL):IF LOF(FIL) = 0 THEN
      ERT = 0:ER = 9:GOSUB 3380 :CLOSE #FIL:
      LP$(LP) = AB$:LP = LP + 1:LP$(LP) = AC$:GOTO 1300
2860  CLOSE #FIL
2870  OPEN"I",FIL,F$(FIL)
2880  PRINT "Processing File ";F$(FIL);CHR$(13);
2890  WHILE EOF(FIL)<> - 1
2900    INPUT #FIL,A$
2910    VL = VAL("&H" + MID$(A$,2,2))
2920    PK = VAL("&H" + MID$(A$,4,4))
2930    IF PK1 = 0 THEN PK1 = PK:PK2 = PK + VL - 1
2940    IF PK - PK2>1 THEN 3080
2950    IF VL = 0 THEN A$ = " ":GOTO 3080
2960    PK2 = PK + VL - 1
2970    B$ = MID$(STR$(LN),2) + " DATA "
2980    FOR I = 1 TO VL
2990      B$ = B$ + MID$(STR$(VAL("&H" + MID$(A$,8 + I*2,2))),2) + '
' , "
3000    NEXT
3010    B$ = LEFT$(B$,LEN(B$) - 1)
3020    PRINT #1,B$
3030    LN = LN + IN
3040  WEND
3050  CLOSE #FIL
3060  FIL = FIL - 1
3070  RETURN
3080  B$ = MID$(STR$(LN),2) + " FOR HEX = " + MID$(STR$(PK1),2) +
      "TO " + MID$(STR$(PK2),2)
3090  PRINT #1,B$
3100  LN = LN + IN
3110  B$ = MID$(STR$(LN),2) + CHR$(9) + "READ HEX1"
3120  PRINT #1,B$
3130  LN = LN + IN
3140  B$ = MID$(STR$(LN),2) + CHR$(9) + "POKE HEX,HEX1"
3150  PRINT #1,B$
3160  LN = LN + IN
3170  B$ = MID$(STR$(LN),2) + " NEXT"
3180  PRINT #1,B$
3190  LN = LN + IN
3200  PK1 = PK:PK2 = PK + VL - 1
3210  IF VL = 0 THEN 3050
3220  GOTO 2970
3230  '---------- FIND LABEL BINARY SEARCH ----------
3240  TOP = LIN:BOT = 1
3250  LOOK = INT((TOP + BOT)/2)
3260  IF A$ = LBL$(LOOK) THEN FLAG = LOOK:RETURN
3270  IF A$>LBL$(LOOK) THEN BOT = LOOK + 1:IF BOT<=TOP
      THEN 3250 ELSE FLAG = 0:RETURN
3280  TOP = LOOK - 1:IF TOP>=BOT THEN 3250
3290  FLAG = 0:RETURN
3300  '---------- NEW FILE NAME ----------
3310  FL = INSTR(2,AA$,"  ")
3320  IF FL = 0 THEN FL = LEN(AA$)
3330  FL$ = MID$(AA$,2,FL - 1)
3340  F$ = FL$:GOSUB 2040
3350  P = INSTR(F$," . "):IF P = 0 THEN F$(FIL) = F$ + ".BBS":ELSE
      F$(FIL) = F$
3360  RETURN
3370  '---------- PRINT ERROR MESSAGE ----------
```

```
3380  AB$ = AA$:IF FLAG<>0 THEN
      AB$ = AB$ + " ' " + LBL$(FLAG) + " is Line #" + STR$(LIN(FLAG))
3390  TB = INSTR(AB$,CHR$(9))
3400  IF TB<>0 THEN MID$(AB$,TB,1) = " ":GOTO 3390
3410  PRINT:PRINT AB$
3420  NERROR = NERROR + 1:ERT = ERT - LEN(STR$(VAL(AB$))) - 2:
      IF ERT<0 THEN ERT = 0
3430  AC$ = MID$(STR$(VAL(AB$) + 1),2) + " ' " + STRING$(ERT,32) +
      "↑ "** ERROR " + ER$(ER) + " ** "
3440  PRINT AC$
3450  RETURN
3460  '---------- CHANGE LINE NUMBER ----------
3470  PNTR = INSTR(4,AA$," ")
3480  LN1 = VAL(MID$(AA$,4,PNTR - 4))
3490  IN1 = VAL(MID$(AA$,PNTR))
3500  IF LN1<LN THEN AA$ = MID$(STR$(LN),2) + " ' " + AA$:ER = 8:
      ERT = LEN(STR$(LN)) + 2:GOSUB 3380:
      LP$(LP) = AB$:LP = LP + 1:LP$(LP) = AC$:LP = LP + 1:RETURN
3510  LN = LN1
3520  IF IN1<>0 THEN IN = IN1:LN = LN-IN
3530  RETURN
```

Listing #2 a PreBasic Test Program

```
[TOP]Print "This is a PreBasic Test"
FOR   I = 1 TO 1000
PRINT   I;CHR$(13);
NEXT
;  This is a comment to be ignored in final code and yet one more
'This next section is a renumber command
     .R 1000 100
'Line number 1000
;  not included and not line number 1100
[TRY__AGAIN]INPUT "Do you want to go (O)n or (B)ack";a$
;  for upper or lower case A$
   IF CHR$(ASC(A$) AND 95) = "B" THEN .TOP
   IF CHR$(ASC(A$) AND 95)<>"O" THEN .TRY__AGAIN
INPUT "How many times";A
;  a parameter passing gosub
   GOSUB .ONWARD[A/]
GOTO .TOP
;  I'm happy to say this is almost the end
[ONWARD B/]FOR I = 1 TO B
   PRINT "So onward we go!"
NEXT
RETURN
END
```

Listing #3 Results from Listing #2 after PreBasic

```
10   PRINT "This is a PreBasic Test"
20   FOR I = 1 TO 1000
30   PRINT I;CHR$(13);
40   NEXT
50   'THIS NEXT SECTION IS A RENUMBER COMMAND
1000   'LINE NUMBER 1000
1100   INPUT "Do you want to go (O)n or (B)ack";A$
1200      IF CHR$(ASC(A$) AND 95) = "B" THEN 20
1300      IF CHR$(ASC(A$) AND 95)<>"O" THEN 1100
1400   INPUT "How many times";A
1500      B = A:GOSUB 1700
1600   GOTO 20
1700   FOR I = 1 TO B
1800      PRINT "So onward we go!"
1900   NEXT
2000   RETURN
2100   END
```