PILOT FOR THE APPLE II MICROCOMPUTER

SPECIAL PROBLEM

Presented to the Faculty of the
Department of Computer Science of the
North Texas State University in Partial
Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Richard George Ellis, B.A., M.A.

Denton, Texas

August, 1979

TABLE OF CONTENTS

Page

Special Problem

PILOT FOR THE APPLE II MICROCOMPUTER

PILOT (Programmed Inquiry, Learning or Teaching) is a simple, conversational language developed in 1969 by John A. Starkweather at the University of California Medical Center in San Francisco. Originally designed for computer assisted instructional needs, PILOT also has been effectively used as an introductory computer language.

The core language consists of approximately eight simple instructions or operators which can be conditionally or unconditionally executed (May & Vann, 1978; Starkweather, 1977; Starkweather & Kamp, 1978; Yob, 1977). In addition to the core instruction set, various language extensions have been implemented by different authors to fulfill local requirements or interests (Hamilton & Scott, 1978; Krieger, 1978).

The PILOT system developed for the Apple II microcomputer consists of two programs, PILOT EDITOR and PILOT DRIVER, which are written in Applesoft and which use the Apple II disk operating system. The PILOT system was designed to facilitate easy authoring and execution of programs written in an extended version of the PILOT language. Due to the memory requirements of the programs and the Apple II disk operating system, the PILOT system described here should be executed on a machine with at least 32k bytes of random access memory.

PILOT EDITOR.

The PILOT EDITOR program accepts program statements written in PILOT, examines each line for correct syntax, and formats all statements in a manner which is acceptable to the PILOT DRIVER program. Six editing commands (LIST, DELETE, RENUMBER, QUIZ, RUN, and DONE) facilitate the editing and review of PILOT programs or lesson files.

The PILOT EDITOR maintains up to three hundred PILOT statements in an internal text buffer for rapid editing and examination. Each statement can contain up to 255 characters. When three hundred PILOT statements have been entered into the text buffer, or when available memory space falls below 1000 bytes, then the contents of the text buffer are stored onto floppy disk. Statements stored on disk remain available to the programmer through use of the editing commands, although editing those statements is appreciably slower due to the speed of the disk accessing routines.

To use the PILCT EDITOR program, enter:

```
RUN PILOT EDITOR
```

The editor initially asks for the name of a PILOT LESSON FILE, and then asks whether the file is NEW or OLD. If the programmer specifies that the named file is NEW, then the editor creates the lesson file and a corresponding lesson control file containing certain information about the lesson file. The lesson file is given the name supplied by the programmer. The lesson control file is given the same name with a trailing C.

```
     example:  TIMES TABLES    - lesson file
               TIMES TABLESC   - lesson control file
```

If the programmer specifies that the named file is OLD, then the editor reads the last portion of the lesson file into the text buffer. (The exact number of lines read into the buffer depends on the current size of the lesson file.)

During execution, the PILOT EDITOR creates, utilizes, and deletes a scratch file named Z>. No other file on disk should be named Z> or the contents will be lost.

PILOT DRIVER.

The PILOT DRIVER program reads and interprets a PILOT program created by the PILOT EDITOR. To run a PILOT program enter:

```
RUN PILOT DRIVER
```

Immediately the question

```
NAME OF PILOT LESSON FILE?
```

is displayed. The user should then enter the name of the PILOT lesson file or program. Execution begins with the first statement of the PILOT program. When execution of the PILOT program is complete, the message

```
DONE
```

is displayed and the standard floating point cursor returns to the monitor.

To execute the specified PILOT program, the driver locates the required file and loads it into the text buffer. If the file is too large to reside in the text buffer all at once, then the driver fetches portions of the program as needed.

PILOT Statement Syntax.

For any PILOT statement to be accepted by the PILOT EDITOR, certain minimum syntactic requirements must be met. The general form of a PILOT statement is:

```
line#  label  op;  object
```

where

line# - a four digit line number

label - optional 1 to 6 character label

op; - any PILOT language operator followed immediately by a required semicolon

object - optional information required by the specific language operator

One or more blanks must separate each element of a PILOT statement. If no label is supplied, the PILOT EDITOR inserts a blank label.

Except where permitted under the descriptions of the language operators, the symbols # ; : , and " cannot be used in PILOT statements. These symbols act as delimiters to PILOT (# ;) or to the Apple computer (: , "). As a rule, additional blanks can be inserted between statement elements, although the PILOT EDITOR removes most of them.

Variables.

Numeric and nonnumeric (string) values may be stored in PILOT variables for use in several types of statements (T, A, M, C, I, SY, SN). Variables are identified by the # sign used as a prefix to any unbroken sequence of symbols.

examples: #NAME
#AGE
#1

A variable can contain either numeric or nonnumeric values. All values of variables are stored as strings. If a variable is used in a numeric application, then the value is converted to a number before use. The same variable can contain either numeric or nonnumeric strings at different times. All variables are global and are known throughout the PILOT program, including any subroutines. At most, 50 variables will be retained by the PILOT DRIVER during program execution.

Response Matching and the Match Flag.

The PILOT language permits the analysis of user responses through a combination of A (accept) and M (match) statements. The system stores a user response in an A statement and attempts to match the user response with any of the correct answers located in the object of the next M (match) statement. If a match is found, then the "match flag" is set to YES.

Otherwise, the match flag is set to NO.

Subsequent statements in the program can be "conditioned" or conditionally executed by the presence of a "Y" or "N" conditioner affixed to the statement's operator. For example:

```
JY;  CORRECT!
TN;  OOPS.  WANT TO TRY AGAIN?
```

Here, a jump to the statement labeled CORRECT! is executed if the previous match was successful. If the match was unsuccessful, then the text message, "OOPS. WANT TO TRY AGAIN?", is displayed.

Core Language Operators

T - Type.

The T operator displays or types the object of the statement on the monitor. The displayed text consists of sequences of any characters not including commas, semicolons, colons, or double quotes.

```
1000  T; THIS IS 'T'YPED EXACTLY!
```

Alternately, the object can include a variable reference, in which case the value of the variable is substituted into the object in place of the variable name.

```
1100  T; #NAME
```

The object can also consist of a mixture of text and variable references. In this case, the variable references are set off from the surrounding text by semicolons.

```
1200  T; #NAME; IS YOUR NAME.
```

The T operator can be conditioned by Y or N.

```
1300  GOOD TY; DISPLAY IF MATCH SUCCEDS.
1400  BAD  TN; DISPLAY IF MATCH FAILS.
```

A - Accept.

The A operator temporarily stops execution of the PILOT program to accept a response from the keyboard. If the object of the statement is blank, then the response is stored in a temporary location and is lost when the next A statement is executed.

```
2000  A;
```

The response may be stored for later use, however, by entering a variable name as the object of the statement.

```
2100  A; #AGE
```

The A operator can be conditioned by Y or N.

If a response contains a comma or a colon, as in

```
YES, I AGREE
```

then only that part of the response prior to the comma is accepted. A response with commas or colons is accepted in its entirety if the response is surrounded by double quotes.

```
"YES, I AGREE."
```

M - Match.

The M operator matches a response given to an A statement against a set of "correct" answers listed in the object of the M statement. The correct answers are separated from one another by semicolons.

```
3000  M; CAT;DOG;MOUSE
```

Leading and trailing semicolons on the match list are optional and can be used to include blanks in the first and last answers.

```
3100  M; ; RAN ; RUN ;
```

If one of the correct answers is located anywhere in the response being matched, then the "match flag" is set to YES. Otherwise, the match flag is set to NO.

A match can also be made to the value of a variable by including the variable name in the match list.

```
3200  M; MY NAME;#NAME
```

Alternately, any response is matched if the object of the M statement is blank.

```
3300  M;
```

J - Jump.

The J operator alters the flow of control or execution by jumping to a statement whose label is located in the object of the J statement.

```
4000  J; START
```

The J operator may be conditioned by Y or N.

R - Remark.

The R operator specifies the object of the R statement to be a nonexecutable remark or comment. Remarks are placed in the PILOT program as reminders to the programmer. During program execution, R statements are ignored.

```
5000  R; THIS IS SOME REMARK!
```

U - Use Subroutine.

The U operator alters the flow of control or execution by using or calling a subroutine whose name or label is located in the object of the U statement.

```
6000  U; SUB1
      .
      .
      .
9000  SUB1 R; START OF SUBROUTINE.
      .
      .
      .
9500  ES;
```

The U operator differs from the J operator in an important way. When the ES (End of Subroutine) operator is encountered, the flow of control automatically returns to the statement following the U statement. Subroutines can call other subroutines but only to a depth of nine. The U operator may be conditioned by Y or N.

ES - End of Subroutine.

The ES operator signals the end of a subroutine. When the ES statement is encountered, the subroutine returns program control to the statement following the U statement from which the subroutine was called.

```
7000  ES;
```

A statement object, if present, is ignored and can be used for commentary. The ES statement can be conditioned by Y or N.

E - End of Program.

The E operator stops execution of the PILCT program.

```
8000  E;
```

A statement object, if present, is ignored.

C - Compute.

The C operator performs simple computations on stored variables and constants. Computations can be simple assignments,

```
9000  C; #VAR = 9
9100  C; #VAR = #XVAL
```

or the computations can involve two operands:

```
9200  C; #VAR = quantity + quantity
9300  C; #VAR = quantity - quantity
9400  C; #VAR = quantity * quantity
```

```
9500  C; #VAR = quantity / quantity
9600  C; #VAR = quantity   quantity
```

where 'quantity' refers to either a constant or a variable name. Variables containing nonnumeric values can be specified as quantities in C statements. If such a variable is a target variable (on the left of the equal sign) then the result of the computation is stored in that variable. If a variable on the right of the equal sign contains a nonnumeric value, then a zero is substituted for the value of that variable in the computation while the actual value of the variable remains unchanged.

Blanks must separate all elements and operators of a C statement. The C operator can be conditioned by Y or N.

Local Extensions to the Core Language Operators

SY - Set Yes.

The SY operator sets the match flag to YES if two numeric quantities specified in the object of the SY statement are equal. The quantity to the left of the equal sign must be stored in a variable.

```
1500  SY; #COUNT = 10
1600  SY; #COUNT = #TIMES
```

SN - Set No.

The SN operator is similar to the SY operator. It sets the match flag to NO if the two numeric quantities specified in the object of the SN statement are equal.

```
2500  SN; #TIMES = 18
2600  SN; #LOOPS = #STARTS
```

At least one blank must separate each quantity or variable from the equal sign.

I - Initialize.

The I operator initializes a target variable to a literal string or quantity specified in the object of the I statement. At least one blank must separate the target variable from the assignment operator, and the assignment operator from from the literal value.

```
3500  I; #SCOLON = ;
3600  I; #NEW = OLD
```

The I operator can be conditioned by Y or N.

P - Pause.

The P operator temporarily halts execution of the PILOT program and displays the message

```
PRESS RETURN
```

Execution of the program continues when the RETURN key (carriage return) is pressed.

```
4500  P;
```

A statement object, if present, is ignored.

D - Display.

The D operator changes the speed at which the characters of the PILOT program are displayed. Speed values are from 25 to 255 and are specified in the object of the D statement.

```
5500  D; 200
```

If a speed outside the proper range is specified, then the display speed defaults to 255.

B - Blank Screen.

The B operator blanks the monitor screen, erasing all currently displayed characters.

```
6500  B;
```

A statement object, if present, is ignored.

G - Get Program.

The G operator chains to another PILOT program specified by name in the object of the G statement. Optionally, a line number can be specified indicating at which line execution of the new program is to begin.

```
7500  G; NEWPROG
7600  G; PROG#2 5000
```

A blank must separate the name of the program from the line number. If no line number is specified, execution of the new program begins with the first statement in the new program. Values of variables stored in the old program are preserved and are available for use in the new program. The G operator can be conditioned by Y or N.

Editing Commands

The following editing commands are available to programmers using the PILOT EDITOR program to create PILOT lesson files or programs.

LIST.

The LIST command displays part or all of the current PILOT program. Execution of the LIST command generates a system query as to the desired output speed. An output speed between 25 and 255 must be specified or a default of 255 is used. Two forms of the LIST command are available.

```
(a)  LIST
(b)  LIST xxxx-yyyy
```

Form (a) lists the entire PILOT program. Form (b) lists statements between line number xxxx and line number yyyy, inclusive. If xxxx equals yyyy, then only a single line of program is listed. The value of xxxx cannot exceed yyyy.

```
examples:   LIST
            LIST 1000-3300
            LIST 1450-1450
```

DELETE.

The DELETE command deletes a specified range of program statements. The form of the DELETE command is:

```
DELETE xxxx-yyyy
```

All statements between and including statements numbered xxxx and yyyy are deleted from the PILOT program file. Both xxxx and yyyy must be included in the command. While xxxx can equal yyyy, in which case, only a single line is deleted, xxxx cannot exceed yyyy.

```
examples:   DELETE 1500-1700
            DELETE 2250-2250
```

RENUMBER.

The RENUMBER command renumbers all PILOT program statements using an initial value of 2000 and an increment specified by the command. The general form is:

```
RENUMBER x
```

where x is the increment. The increment can be any positive whole number between 1 and 7999.

```
examples:   RENUMBER 10
            RENUMBER 540
```

QUIZ.

The QUIZ command displays certain information about the current PILOT program. Included are:

(a) the number of statements in the PILOT program file

(b) the lowest and highest line numbers in the program

(c) the number of statements in the text buffer

(d) the lowest and highest line numbers in the buffer

(e) the number of unused bytes of memory

example: QUIZ

DONE.

The DONE command 1) saves onto disk that part of the PILOT program file still in the text buffer, 2) issues the message:

FILE SAVED

and 3) stops execution of the PILOT EDITOR.

example: DONE

RUN.

The RUN command saves onto disk that part of the PILOT program file in the text buffer, and then loads and executes the PILOT DRIVER.

example: RUN

Statement Insertion and Replacement.

PILOT statements can be entered in any order. The PILOT EDITOR maintains all statements in the correct order by line number. Statements can be inserted into the program file by supplying an appropriate line number. A statement can replace another statement with the same line number merely by entering the new statement. A single statement can be deleted only by using the DELETE command.

System Design Characteristics

Text Lesson File and Text Control File.

Both the PILOT EDITOR and the PILOT DRIVER require two disk files. The text lesson file (TLF$) is a sequential file containing a series of strings representing the lines of a PILOT program. The text control file (TCF$) is a sequential file containing the program count (PC), the program low line number (PL$) and the program high line number (PH$), for the corresponding text lesson file. These files are created, referenced, and updated as necessary during execution of the PILOT EDITOR. During execution of the PILOT DRIVER, the two files are referenced only.

PILOT EDITOR.

The PILOT EDITOR requires approximately 6k bytes of memory excluding memory requirements for the PILOT program residing in the text buffer (TB$). The editor accepts input from the keyboard, one line at a time. Each line is examined for the presence of a command (LIST, RENUMBER, DELETE, QUIZ, DONE, RUN). If a command is found, transfer is made to a processing routine which handles the request. Otherwise, the input line is analyzed as a PILOT instruction. If the PILOT instruction is acceptable (proper syntax), then the statement is formatted and inserted into the next element of the text buffer.

If an error is detected either in a command or in a PILOT statement, then an error message is displayed and the editor is readied to accept another line from the keyboard without further processing. (See Appendix B for editor error messages.)

The PILOT EDITOR will continue to accept PILOT statements into the text buffer until one of three situations occurs. 1) The user terminates the editing session by issuing a DONE or RUN command. 2) The number of PILOT statements reaches 300, the maximum held by the array TB$. In this case, the contents of the text buffer will be stored onto disk and the text buffer will be effectively emptied. 3) The available memory left in the computer falls below 1000 bytes, in which case the text buffer is emptied onto disk as in case 2. All editing commands and facilities are applicable over the entire PILOT lesson file, including those statements stored on disk. However, editing statements stored on disk is time-consuming due to the speed of the disk accessing routines. It is recommended that all PILOT lesson files be created with less than the maximum number of lines to speed both editing and execution of the PILOT program. Multiple segments of one large lesson can be "chained" together using the G statement to gain the benefits of both fast editing and execution, and lengthy instructional sequences.

The major software function modules and locations are as follows:

1. Initialization of variables (10 - 890).
2. Text line input and analysis (1000 - 1020).

3. Analysis of text buffer extent and optional text transfer to disk (1500)
4. PILOT statement syntax check (2000 - 2280)
5. Insertion of PILOT statement into text file (2500 - 2570).
6. Subroutine - Extraction of the next sequence of characters from the line of text (LT$) (8000 - 8040).
7. Subroutine - Insertion of PILOT statement into text buffer (8300 - 8460).
8. LIST command processing (10000 - 10160).
9. RENUMBER command processing (11000 - 11070).
10. DELETE command processing (12000 - 12600).
11. RUN command processing (13000).
12. QUIZ command processing (14000 - 14050).
13. Subroutine - Transfer of text buffer onto disk (18000 - 18050).
14. Subroutine - Update of PILOT text control file variables PC, PL$, and PH$ (18100).
15. Subroutine - Initialization of PILOT text control variables PC, PL$, and PH$ from text control file (18150).
16. Subroutine - Loading of next block of PILOT text file statements (up to 300) into the text buffer (18200 - 18260).
17. Subroutine - Loading of last full or partial block of text file statements into text buffer (18300 - 18340).

18. Subroutine - Transfer of text buffer onto disk in scratch file Z (18400 - 18420).
19. DONE command processing (25000).

PILOT DRIVER.

The PILOT DRIVER requires approximately 4k bytes of memory excluding memory requirements of the PILOT program residing in the text buffer. When a PILOT program is executed, the first block (usually equal to 300 statements or less) is loaded into the text buffer. All text lines are scanned for labels which, if found, are entered into a jump table (JT$) to speed processing of J and U commands. During execution, subroutine return addresses are stacked in ST$. Variables are stored in the variable table (VT$).

PILOT text lines are interpreted and executed sequentially according to line number unless the flow of control is altered with a J, U, or ES command. Program execution terminates when an E instruction is encountered.

If the PILOT lesson file exceeds 300 lines or so in length, then the PILOT DRIVER fetches and executes successive blocks of text as needed. The disk accessing routines and the initialization of the jump table are time consuming processes which significantly interrupt the flow of instruction from the PILOT program. Therefore, it is recommended that lessons be constructed in such a manner that each unit consists of 300 or fewer lines of PILOT statements.

Major software function modules and locations are as follows:

1. Initialization of variables (10 - 900).
2. Subroutine - Extraction of the next sequence of characters from a PILOT statement (LT$) (8000 - 8040).
3. Subroutine - Jump table access (9000 - 9060).
4. Subroutine - Variable table look up and insertion (10000 - 10050).
5. Initialization of variables (15000).
6. Program counter increment and text lesson file access (15005 - 15010).
7. Statement type analysis and branch (15030 - 16000).
8. Processing I, IY, IN instructions (16100).
9. Processing SY, SN instructions (16300 - 16350).
10. Processing C, CY, CN instructions (16400 - 16540).
11. Processing M, MY, MN instructions (16700-16820).
12. Processing A, AY, AN instructions (16900 - 16940).
13. Processing ES, ESY, ESN instructions (17100 - 17110).
14. Processing U, UY, UN instructions (17260 - 17270).
15. Processing J, JY, JN instructions (17290).
16. Processing T, TY, TN instructions (17400 - 17490).
17. Processing G, GY, GN instructions (17500 - 17520).
18. Processing D instruction (17600 - 17620).
19. Subroutine - Initialization of text control variables PC, PL$, and PH$ from text control file (18150).
20. Subroutine - Loading of next block of PILOT text file statements (up to 300) into the text buffer

(18200 - 18260).

21. Subroutine - Initialization of Jump table (19100 - 19160).

Appendix A

Major Variables

Variable	Use
ST$(10)	Subroutine return address stack
TB$(300)	PILOT text buffer
JT$(300,2)	Jump label table
VT$(50,2)	Variable/value table
TLF$	PILOT text lesson file
TCF$	PILOT text control file
Z >	Scratch file
P2	Program counter
FP	PILOT text lesson file pointer
FSP	Scratch file pointer
PC	Program count
PL$	Program Low line number
PH$	Program High line number
BC	Buffer count
BL$	Buffer Low line number
BH$	Buffer High line number
LT$	Line of text
D$	Control D - required by DOS

Appendix B

PILOT EDITOR Error Messages

1. ERROR-NO LINE # - no line number in PILOT statement
2. ERROR-LINE # OUT OF BOUNDS - illegal line number given in PILOT statement
3. ERROR-NO OPERATOR - no semicolon found after the operator
4. ERROR-LABEL TOO LONG - more than six characters given as the label of a PILOT statement
5. ERROR-JUMP LABEL TOO LONG - more than six characters specified in the object of a J or U statement
6. ERROR-NO TARGET VARIABLE - a proper variable was not specified to the left of the equal sign in the object of a C, SY, SN, or I statement
7. ERROR-NO '=' SIGN - the equal sign is missing in a C, SY, SN, or I statement
8. ERROR-ILLEGAL OPERATOR - an invalid arithmetic binary operation was specified in a C statement
9. ERROR-BAD OPERATOR - an illegal PILOT language operator was specified in a PILOT statement
10. ERROR-BAD COMMAND - an editing command was issued with improper syntax
11. NO PROGRAM - a LIST command was issued, but no program exists

Appendix C

Fatal Execution Time Errors

1. PROGRAM ERROR-BAD JUMP EXECUTED FROM LINE #

2. PROGRAM ERROR-DIVIDE BY ZERO IN LINE #

3. PROGRAM ERROR-RETURN WITHOUT SUBROUTINE CALL IN LINE #

4. PROGRAM ERROR-SUBROUTINES NESTED TOO DEEPLY IN LINE #

References

Hamilton, R. L., & Scott, D. W. A new approach to computer assisted instruction in music theory. *Proceedings of the 9th Conference for Computers in the Undergraduate Curriculum*, 1978.

Krieger, A. S. Programmed instruction made easy: Tiny PILOT. *Kilobaud*, 1978, (16), 70-77.

May, R., & Vann, K. A programming language for beginners. *Interface Age*, 1978, *3*(9), 64-67.

Starkweather, J. A. *Guide to 8080 PILOT, version 1.1*. Report prepared for the Lister Hill National Center for Biomedical Communications, Bethesda, Md., 1977. (NTIS No. PB 270 715)

Starkweather, J. A., & Kamp, M. A self-contained CAI machine for health sciences education. In Edward C. Deland (Ed.), *Information Technology in Health Science Education*. New York: Plenum Press, 1978.

Yob, G. PILOT. *Creative Computing*, 1977, *3*(3), 57-63.

```
]LIST
10  REM PILOT EDITOR COPYRIGHT (C), 1979 BY RICHARD G. ELLIS
100 FC = 300
400  DIM TB$(FC),ST$(10),VT$(50,2)
450 P1 = 0:P2 = P1:P3 = P1:P4 = P1:F1 = P1
500 D$ = "":B$ = " ":C0 = 0:C1 = 1:C2 = 2:C3 = 3:C4 = 4:C5 = 5:C6 = 6:C8 =
     8:CT = 1000:CZ$ = "0000":CMD$ = "LISRENDELRUNQUIDONTAT":PC = C0:PL$ =
     CZ$:PH$ = CZ$:BC = C0:BL$ = PL$:BH$ = PH$
800  INPUT "NAME OF PILOT LESSON FILE?";TLF$: PRINT D$;"NOMON C,I,O"
810  IF TLF$ = "" THEN 800
820 TCF$ = TLF$ + "C": PRINT "IS ";TLF$;" NEW OR OLD?": INPUT M$
830  IF M$ = "OLD" THEN  GOSUB 18150: GOSUB 18300: GOTO 1000
840  IF M$ < > "NEW" THEN 820
860  PRINT D$;"OPEN " + TCF$: PRINT D$;"WRITE " + TCF$
880  PRINT PC: PRINT PL$: PRINT PH$
900  PRINT D$;"CLOSE " + TCF$
1000  INPUT "->";LT$:PA$ =  LEFT$ (LT$,C3):P1 = C1: REM  ENTER
      TEXT LINE
1010  IF P1 <  = C6 AND PA$ <  >  MID$ (CMD$,(P1 - C1) * C3 + C1,C3) THEN
     P1 = P1 + C1: GOTO 1010
1020  ON P1 GOTO 10000,11000,12000,13000,14000,25000
1500  IF BC >  = FC OR  ABS ( FRE (C0)) < CT THEN  GOSUB 18000: REM SAVE B
     UFFER
2000  REM CHECK SYNTAX
2010  GOSUB 8000:LN =  VAL (PA$):LN$ = PA$:LA$ = "      ": GOSUB 8000
2020  IF  NOT LN THEN  PRINT "ERROR-NO LINE #": GOTO 1000
2030  IF LN < CT1 OR LN > 9999 OR  INT (LN) <  > LN THEN  PRINT "ERROR-LIN
     E # OUT OF BOUNDS": GOTO 1000
2040  IF PA$ = B$ THEN  PRINT "ERROR-NO OPERATOR": GOTO 1000
2050  IF  RIGHT$ (PA$,C1) = "," THEN 2090
2060 LA$ = PA$: GOSUB 8000
2070  IF  LEN (LA$) > C6 THEN  PRINT "ERROR-LABEL TOO LONG": GOTO 1000
2080  IF  LEN (LA$) < C6 THEN LA$ = LA$ + B$: GOTO 2080
2090 OP$ = PA$:OB$ = LT$
2095 C$ =  LEFT$ (OP$,C1)
2100  IF C$ <  > "J" AND C$ <  > "U" THEN 2150
2110  GOSUB 8000
2120  IF  LEN (PA$) > C6 THEN  PRINT "ERROR-JUMP LABEL TOO LONG": GOTO 100
     0
2130  IF  LEN (PA$) < C6 THEN PA$ = PA$ + B$: GOTO 2130
2140 OB$ = PA$: GOTO 2240
2150  IF C$ <  > "C" AND C$ <  > "S" THEN 2233
2160 OB$ = "":P2 = C0
2170 P2 = P2 + C1
2180  IF LT$ = B$ THEN 2231
2190  GOSUB 8000:OB$ = OB$ + B$ + PA$
2200  IF P2 = C1 AND  LEFT$ (PA$,C1) <  > "#" THEN  PRINT "ERROR-NO TARGET
      VARIABLE": GOTO 1000
2210  IF P2 = C2 AND PA$ <  > "=" THEN  PRINT "ERROR-NO  =  SIGN": GOTO 10
     00
2220  IF P2 = C4 AND  NOT (PA$ = "+" OR PA$ = "-" OR PA$ = "*" OR PA$ = "/
     " OR PA$ = "^") THEN  PRINT "ERROR-ILLEGAL OPERATOR": GOTO 1000
2230  GOTO 2170
2231 OB$ =  MID$ (OB$,C2)
2232  IF C$ <  > "1" THEN 2240
2234  GOSUB 8000
2235  IF  LEFT$ (PA$,C1) <  > "#" THEN  PRINT "ERROR-NO TARGET VARIABLE": GOTO
     1000
2236  GOSUB 8000
2237  IF PA$ <  > "=" THEN  PRINT "ERROR-NO  =  SIGN": GOTO 1000
2240  IF OP$ = "T, " OR OP$ = "TY, " OR OP$ = "TN, " OR OP$ = "M, " OR OP$ = "
     MY, " OR OP$ = "MN, " OR OP$ = "A, " OR OP$ = "AY, " OR OP$ = "AN, " OR OP
     $ = "J, " OR OP$ = "JY, " OR OP$ = "JN, " THEN 2280
      IF OP$ = "CY, " OR OP$ = "CN, " OR OP$ = "R, " OR OP$ = "R: " OR OP$ = "
     "IY," OR OP$ = "IN," THEN 2280
2250  IF OP$ = "C," OR OP$ = "CY," OR OP$ = "CN," OR OP$ = "U," OR OP$ = "
     UY," OR OP$ = "UN," OR OP$ = "E," OR OP$ = "ES," OR OP$ = "ESY," OR O
     P$ = "ESN," OR OP$ = "R," THEN 2280
2260  PRINT "ERROR-BAD OPERATOR". GOTO 1000
2280 LT$ = LN$ + B$ + LA$ + B$ + OP$ + B$ + OB$
2500  REM INSERT LINE INTO BUFFER
2505  IF LN$ <  = PH$ AND BC = C0 THEN 2520
2510  IF LN$ >  = BL$ THEN  GOSUB 8300. GOTO 1000
2520  GOSUB 18000:FP1 = C0.FSP = C0:FC = FC - 1:TL = FC
2530  GOSUB 18200
2540  IF LN$ >  = BL$ AND LN$ <  = BH$ THEN  GOSUB 8300.FC = FC + C1
2550  GOSUB 18400
2560  IF FSP < FC THEN 2530
2570  PRINT D$,"DELETE " + TLF$. PRINT D$,"RENAME 2.," + TLF$: GOSUB 18100
      . GOSUB 18300: GOTO 1000
8000  REM DETACH A WORD FROM LT$
8010 P1 = C1: IF LT$ = B$ OR LT$ = "" THEN PH$ = B$: RETURN
8020  IF  LEFT$ (LT$,C1) = B$ THEN LT$ =  MID$ (LT$,C2). GOTO 8020
8030  IF  MID$ (LT$,P1,C1) = B$ THEN 8040
8035 P1 = P1 + C1: IF P1 >  LEN (LT$) THEN PH$ = LT$.LT$ = B$: RETURN
8037  GOTO 8030
8040 PH$ =  LEFT$ (LT$,P1 - C1).LT$ =  MID$ (LT$,P1 + C1). RETURN
8300  REM INSERT LINE INTO BUFFER
8310 FI = C0.P1 = BC
8315  IF LN$ > PH$ THEN PH$ = LN$
8316  IF PL$ = C2$ THEN PL$ = LN$
8317  IF LN$ < PL$ THEN PL$ = LN$
8318  IF  NOT P1 THEN BL$ = LN$
8320  IF LN$ > BH$ THEN TB$(P1 + C1) = LT$.BC = BC + 1.FI = C1.BH$ = LN$:P
     C = PC + C1. RETURN
8340 P1 = C1
8350  IF P1 > BC OR FI = C1 THEN P1 = BC: GOTO 8400
8360  REM REPLACE LINE WITH SAME NUMBER
8370 L2$ =  LEFT$ (TB$(P1),4)
8380  IF L2$ = LN$ THEN TB$(P1) = LT$.FI = C1. GOTO 8350
8390 P1 = P1 + C1: GOTO 8350
8400  IF FI THEN  RETURN
8410  REM RELOCATE CURRENT LINES AND INSERT
8420 TB$(P1 + C1) = TB$(P1).P1 = P1 - C1
8430  IF P1 > C1 THEN  GOTO 8440
8432 L2$ =  LEFT$ (TB$(P1),C4)
8434  IF LN$ > L2$ THEN TB$(P1 + C1) = LT$
8436  IF LN$ < L2$ THEN TB$(P1 + C1) = TB$(P1).TB$(P1) = LT$.BL$ = LN$
8438 BC = BC + 1.FI = C1. GOTO 8400
8440 L2$ =  LEFT$ (TB$(P1),C4)
8450  IF LN$ > L2$ THEN TB$(P1 + C1) = LT$.BC = BC + 1.FI = C1:PC = PC + C
     1
8460  GOTO 8400
10000  REM LIST COMMAND
10020  IF  RIGHT$ (LT$,C1) = B$ THEN LT$ =  LEFT$ (LT$, LEN (LT$) - C1). GOTO
     10020
10030 P1 =  LEN (LT$)
10040  IF P1 < C5 THEN LL$ = PL$.LH$ = PH$: GOTO 10085
10050  IF P1 = 14 THEN 10070
10060  PRINT "ERROR-BAD COMMAND": GOTO 1000
10070 LL$ =  MID$ (LT$,C6,C4).LH$ =  RIGHT$ (LT$,C4).P1 =  VAL (LL$).P2 =
      VAL (LH$)
10075  IF P1 > P2 THEN 10060
10080  IF P1 < CT1 OR P2 < C1 THEN 10060
10085  INPUT "SPEED OF OUTPUT? ",P2
10087  IF P2 < 10 OR P2 > 255 THEN P2 = 255
10088 P1 = C1
```

```
10120  IF P1 > BC THEN  SPEED= 255: GOSUB 18200: SPEED= P2:P1 = C1
10130 LN$ =  LEFT$ (TB$(P1),C4)
10140  IF LN$ >  = LL$ AND LN$ <  = LH$ THEN  PRINT TB$(P1)
10150  IF LN$ = PH$ THEN  SPEED= 255: GOTO 1000
10160 P1 = P1 + C1: GOTO 10120
10999  REM RENUMBER COMMAND
11000 LC = 2000:LT$ = LT$ + "E":L2 =  INT ( VAL ( MID$ (LT$,10))):P1 = L2 *
     PC + LC
11010  IF P1 <  = LC OR P1 > 9999 THEN  PRINT "ERROR-BAD COMMAND": GOTO 10
     00
11020  GOSUB 18000:FP1 = C0.FSP = C0:TL = PC
11030  GOSUB 18200
11040  FOR P1 = C1 TO BC.TB$(P1) =  STR$ (LC) +  MID$ (TB$(P1),C5).LC = LC
      + L2: NEXT
11050  GOSUB 18400
11060  IF FSP < PC THEN 11030
11070  PRINT D$;"DELETE " + TLF$. PRINT D$;"RENAME Z>," + TLF$.PL$ = "2000
     ":PH$ =  STR$ (LC - L2).BL$ =  LEFT$ (TB$(C1),C4).BH$ = PH$. GOSUB 18
     100: GOSUB 18300: GOTO 1000
11999  REM DELETE COMMAND
12000 LT$ = LT$ + "E":LL$ =  MID$ (LT$,C8,C4):LH$ =  MID$ (LT$,13,C4)
12010  IF  VAL (LL$) < CT1 OR  VAL (LH$) < CT THEN  PRINT "ERROR-BAD COMMA
     ND": GOTO 1000
12012  IF LH$ > LL$ THEN  PRINT "ERROR-BAD COMMAND". GOTO 1000
12015  IF LL$ >  = BL$ THEN  GOSUB 12500. GOTO 1000
12020  GOSUB 18000:FP1 = C0:FSP = C0:TL = PC
12030  GOSUB 18200
12040  IF BH$ < LL$ OR BL$ > LH$ GOTO 12060
12050  GOSUB 12500
12060  GOSUB 18400
12070  IF FSP < PC THEN 12030
12080  PRINT D$;"DELETE " + TLF$: PRINT D$;"RENAME Z>," + TLF$:PH$ =  LEFT$
     (TB$(BC),C4):BH$ = PH$:BL$ =  LEFT$ (TB$(C1),C4): GOSUB 18100: GOSUB
     18300: GOTO 1000
12500  REM DELETE FROM BUFFER
12510 P2 = BC:P1 = C0
12520  FOR P3 = C1 TO P2
12530 LN$ =  LEFT$ (TB$(P3),C4)
12540  IF LN$ < LL$ THEN P1 = P3: GOTO 12560
12550  IF LN$ > LH$ THEN P1 = P1 + C1:TB$(P1) = TB$(P3)
12555  IF LN$ >  = LL$ AND LN$ <  = LH$ THEN BC = BC - C1.PC = PC - C1
12560  NEXT P3
12570  IF BL$ = PL$ THEN PL$ =  LEFT$ (TB$(C1),C4)
12580 BL$ =  LEFT$ (TB$(C1),C4):BH$ =  LEFT$ (TB$(BC),C4)
12590  IF PH$ <  = LH$ THEN PH$ = BH$
12600  RETURN
13000  GOSUB 18000: PRINT D$;"RUN PILOT DRIVER": REM RUN COMMAND
14000  REM QUIZ COMMAND
14010  PRINT : PRINT "THERE ARE ";PC;" LINES IN THE LESSON FILE."
14020  PRINT "LINE NUMBERS RANGE FROM ";PL$;" TO ";PH$;"."
14030  PRINT "THERE ARE ";BC;" LINES IN THE BUFFER."
14040  PRINT "LINES ";BL$;" THRU ";BH$;" ARE IN THE BUFFER."
14050  PRINT "THE REMAINING FREE SPACE = "; FRE (0);" BYTES.": GOTO 1000
17600 TP = C0. REM TABLE LOOK-UP
17610 TP = TP + C1
17620  IF TP > TC THEN VT$(TP,C1) = V$:VT$(TP,C2) = M$:TC = TC + C1: RETURN

17630  IF VT$(TP,C1) <  > V$ THEN 17610
17640  IF TS THEN M$ = VT$(TP,C2): RETURN
17650 VT$(TP,C2) = M$: RETURN
18000 P3 = C0.FP1 = FC
18010  IF FP1 > FC THEN FP1 = FP1 - FC:P3 = P3 + FC: GOTO 18010
18020  PRINT D$;"OPEN " + TLF$. PRINT D$;"POSITION " + TLF$ + " ,R";P3: PRINT
     D$;"WRITE " + TLF$
18030  IF BC = C0 THEN 18050
```

```
040  FOR P3 = C1 TO BC: PRINT TB$(P3): NEXT
050  PRINT D$;"CLOSE " + TLF$: GOSUB 18100:BC = C0:BL$ = C2$:BH$ = C2$: RETUR
100  PRINT D$;"OPEN " + TLF$: PRINT D$;"WRITE " + TLF$: PRINT FC: PRINT
    PL$: PRINT PH$: PRINT D$;"CLOSE " + TCF$: RETURN
150  PRINT D$;"OPEN " + TCF$: PRINT D$;"READ " + TLF$: INPUT FC: INPUT P
    L$: INPUT PH$: PRINT D$;"CLOSE " + TCF$: RETURN
195  REM NEXT BLOCK INTO BUFFER
200  PRINT D$;"OPEN " + TLF$: PRINT D$;"POSITION " + TLF$ + " ,R";FP1: PRINT
    D$;"READ " + TLF$
210  FOR P3 = C1 TO FC: INPUT TB$(P3):FP1 = FP1 + C1
220  IF FP1 = TL THEN BC = P3: GOTO 18260
230  IF  ABS ( FRE (C0)) < CT THEN BC = P3: GOTO 18260
240  NEXT
250 BC = FC
260  PRINT D$;"CLOSE " + TLF$:BL$ =  LEFT$ (TB$(C1),C4):BH$ =  LEFT$ (TB
    $(BC),C4): RETURN
295  REM LAST BLOCK INTO BUFFER
300 P3 = C0:FP1 = PC
310  IF FP1 > FC THEN FP1 = FP1 - FC:P3 = P3 + FC: GOTO 18310
320  PRINT D$;"OPEN " + TLF$: PRINT D$;"POSITION " + TLF$ + " ,R";P3: PRINT
    D$;"READ " + TLF$
325 BC = FP1
330  FOR P3 = C1 TO FP1: INPUT TB$(P3): NEXT
340  PRINT D$;"CLOSE " + TLF$:BC = FP1:BL$ =  LEFT$ (TB$(C1),C4):BH$ =  LEFT$
    (TB$(BC),C4): RETURN
395  REM WRITE BUFFER TO SCRATCH
400  PRINT D$;"OPEN Z>": PRINT D$;"POSITION Z>,R";FSP: PRINT D$;"WRITE Z
    >"
410  FOR P1 = C1 TO BC: PRINT TB$(P1): NEXT
420  PRINT D$;"CLOSE Z>":FSP = FSP + BC: RETURN
600  GOSUB 18000: PRINT "FILE SAVED": END
```

```
17099  REM PROCESS ES,ESY,ESN
17100  IF  NOT SP THEN  PRINT "PROGRAM ERROR-RETURN WITHOUT SUBROUTINE CAL
      L IN LINE #";LN$: END
17110 OB$ = ST$(SP):SP = SP - C1: GOSUB 9000: GOTO 15005
17259  REM PROCESS U,UY,UN
17260 SP = SP + C1:ST$(SP) = LN$
17270  IF SP > SM THEN  PRINT "PROGRAM ERROR-SUBROUTINES NESTED TOO DEEPLY
       IN LINE #";LN$: END
17289  REM PROCESS J,JY,JN
17290 OB$ =  MID$ (LT$,C1): GOSUB 9000: GOTO 15010
17399  REM PROCESS T,TY,TN
17400 OB$ = LT$ + ",":LT$ = "":P1 = C1:P4 =  LEN (OB$):FI = C0:M$ = B$:TS =
      C1
17410  FOR P3 = C1 TO P4:C$ =  MID$ (OB$,P3,C1)
17420  IF C$ = "#" THEN P1 = P3:FI = C1: GOTO 17480
17430  IF C$ <  > "," THEN 17480
17440  IF  NOT FI THEN LT$ = LT$ +  MID$ (OB$,P1,P3 - P1):P1 = P3 + C1: GOTO
      17480
17450 V$ =  MID$ (OB$,P1,P3 - P1)
17460  IF  RIGHT$ (V$,C1) = B$ THEN V$ =  LEFT$ (V$, LEN (V$) - C1): GOTO
      17460
17470  GOSUB 10000:LT$ = LT$ + M$:P1 = P3 + C1:FI = C0
17480  NEXT
17490  PRINT LT$: GOTO 15005
17499  REM PROCESS G,GY,GN
17500  GOSUB 8000:TLF$ = PH$: GOSUB 8000:LL$ = PH$
17510  IF LL$ = B$ THEN LL$ = "0000"
17520  GOTO 500
17599  REM PROCESS D
17600  GOSUB 8000:P1 =  VAL (PH$)
17610  IF P1 < 25 OR P1 > 255 THEN P1 = 255
17620  SPEED= P1: GOTO 15005
18149  REM READ PC,PL$,PH$ FROM TCF$
18150  PRINT D$;"OPEN " + TCF$: PRINT D$;"READ " + TCF$: INPUT PC: INPUT P
      L$: INPUT PH$: PRINT D$;"CLOSE " + TCF$: RETURN
18199  REM READ IN NEXT BLOCK FROM TLF$
18200  PRINT D$;"OPEN " + TLF$: PRINT D$;"POSITION " + TLF$ + " ,R";FP1: PRINT
      D$;"READ " + TLF$
18210  FOR P3 = C1 TO PC: INPUT TB$(P3):FP1 = FP1 + C1
18220  IF FP1 = PC THEN BC = P3: GOTO 18260
18240  NEXT
18250 BC = PC
18260  PRINT D$;"CLOSE " + TLF$:BL$ =  LEFT$ (TB$(C1),C4):BH$ =  LEFT$ (TB
      $(BC),C4)
18300 JC = C0: REM BUILD JUMP TABLE
18310  IF  NOT BC THEN  RETURN
18320  FOR P1 = C1 TO BC:LA$ =  MID$ (TB$(P1),C6,C6)
18340  IF  LEFT$ (LA$,C1) <  > B$ THEN JC = JC + C1:JT$(JC,C1) = LA$:JT$(J
      C,C2) =  STR$ (P1)
18350  NEXT
18360  RETURN
```

```
15210  IF PA$ = "ESN," THEN 17100
15215  GOTO 15005
15220  IF PA$ = "TY," THEN 17400
15230  IF PA$ = "AY," THEN 16900
15240  IF PA$ = "MY," THEN 16700
15250  IF PA$ = "JY," THEN 17290
15260  IF PA$ = "UY," THEN 17260
15270  IF PA$ = "CY," THEN 16400
15273  IF PA$ = "IY," THEN 16100
15275  IF PA$ = "GY," THEN 17500
15280  IF PA$ = "ESY," THEN 17100
16000  GOTO 15005
16099  REM PROCESS I,IY,IN
16100 TS = C0: GOSUB 8000.V$ = PA$. GOSUB 8000. GOSUB 8000.M$ = PA$. GOSUB
      10000: GOTO 15005
16299  REM PROCESS SY,SN
16300 OP$ = PA$.TS = C1: GOSUB 8000.V$ = PA$  GOSUB 10000.P1 =  VAL (M$):P
      A$ =  MID$ (LT$,C3)
16310  IF  LEFT$ (PA$,C1) = "#" THEN V$ = PA$. GOSUB 10000.P3 =  VAL (M$).
       GOTO 16330
16320 P3 =  VAL (PA$)
16330  IF OP$ = "SY," AND P1 = P3 THEN M = C1
16340  IF OP$ = "SN," AND P1 = P3 THEN M = C0
16350  GOTO 15005
16399  REM PROCESS C,CY,CN
16400 TS = C1.M$ = B$. GOSUB 8000:LA$ = PA$. GOSUB 8000. GOSUB 8000
16410  IF  LEFT$ (PA$,C1) = "#" THEN V$ = PA$. GOSUB 10000.P3 =  VAL (M$).
       GOTO 16430
16420 P3 =  VAL (PA$)
16430  IF LT$ = B$ THEN M$ = PA$.TS = C1.V$ = LA$. GOTO 16540
16440  GOSUB 8000.OP$ = PA$. GOSUB 8000
16450  IF  LEFT$ (PA$,C1) = "#" THEN V$ = PA$. GOSUB 10000.P4 =  VAL (M$):
       GOTO 16470
16460 P4 =  VAL (PA$)
16470 TS = C0.V$ = LA$
16480  IF OP$ = "+" THEN M$ =  STR$ (P3 + P4)
16490  IF OP$ = "-" THEN M$ =  STR$ (P3 - P4)
16500  IF OP$ = "*" THEN M$ =  STR$ (P3 * P4)
16510  IF OP$ = "/" AND  NOT P4 THEN  PRINT "PROGRAM ERROR-DIVIDE BY ZERO
      IN LINE # ";LN$: END
16520  IF OP$ = "/" THEN M$ =  STR$ (P3 / P4)
16530  IF OP$ = "^" THEN M$ =  STR$ (P3 ^ P4)
16540  GOSUB 10000: GOTO 15005
16699  REM PROCESS M,MY,MN
16700 M$ = A$.LN =  LEN (LT$).M = C0
16710  IF LT$ = B$ THEN M = C1. GOTO 15005
16720  IF  RIGHT$ (LT$,C1) <  > "," THEN LT$ = LT$ + ","
16730 P4 =  LEN (M$):M = C0.P1 = C1.PI =  LEN (LT$)
16740  FOR P3 = P1 TO PI
16750  IF  MID$ (LT$,P3,C1) <  > "," THEN 16810
16755  IF P3 = P1 THEN 16800
16760 PA$ =  MID$ (LT$,P1,P3 - P1).PI =  LEN (PA$)
16765  IF  LEFT$ (PA$,C1) = "#" THEN V$ = PA$.LA$ = M$.TS = C1. GOSUB 1000
       0.PA$ = M$.M$ = LA$:PI =  LEN (PA$)
16770  FOR P1 = C1 TO LN - PI + C1
16780  IF  MID$ (M$,P1,PI) = PA$ THEN M = C1. GOTO 15005
16790  NEXT P1
16800 P1 = P3 + C1
16810  NEXT P3
16820  GOTO 15005
16899  REM PROCESS A,AY,AN
16900  INPUT "->?";M$.A$ = M$.TS = C0: GOSUB 8000
16910  IF M$ = "" THEN M$ = B$
16920  IF  LEFT$ (M$,C1) = B$ THEN M$ =  MID$ (M$,C2). GOTO 16920
16930  IF  LEFT$ (PA$,C1) = "#" THEN V$ = PA$. GOSUB 10000
16940  GOTO 15005
```

```
JLIST
10  REM PILOT DRIVER COPYRIGHT (C), 1979 BY RICHARD G. ELLIS
100 FC = 300.TC = C0
400  DIM TB$(FC),ST$(10),VT$(50,2),JT$(FC,2)
500 D$ = "".B$ = " ":C0 = 0.C1 = 1.C2 = 2.C3 = 3.C4 = 4.C6 = 6.CT = 1000
800  INPUT "NAME OF PILOT LESSON FILE?";TLF$. PRINT D$;"NOMON C,I,O"
810  IF TLF$ = "" THEN 800
900 TCF$ = TLF$ + "C": GOTO 15000
8000  REM DETACH A WORD FROM LT$
8010 P1 = C1: IF LT$ = B$ OR LT$ = "" THEN PH$ = B$. RETURN
8020  IF  LEFT$ (LT$,C1) = B$ THEN LT$ =  MID$ (LT$,C2): GOTO 8020
8030  IF  MID$ (LT$,P1,C1) = B$ THEN 8040
8035 P1 = P1 + C1: IF P1 >  LEN (LT$) THEN PH$ = LT$.LT$ = B$. RETURN
8037  GOTO 8030
8040 PH$ =  LEFT$ (LT$,P1 - C1).LT$ =  MID$ (LT$,P1 + C1). RETURN
9000 P3 = C0. REM ACCESS JUMP TABLE
9005  IF  NOT JC THEN 9040
9010  FOR P1 = C1 TO JC
9020  IF JT$(P1,C1) = OB$ THEN P2 =  VAL (JT$(P1,C2)). RETURN
9030  NEXT
9040  IF  NOT P3 THEN FP1 = C0
9050  IF FP1 >  = FC THEN  PRINT "PROGRAM ERROR-BAD JUMP EXECUTED FROM LIN
     E #";LN$. END
9060  GOSUB 18200:P3 = C1: GOTO 9005
10000 TP = C0. REM TABLE LOOK-UP
10010 TP = TP + C1
10020  IF TP > TC THEN VT$(TP,C1) = V$.VT$(TP,C2) = M$.TC = TC + C1: RETURN

10030  IF VT$(TP,C1) <  > V$ THEN 10010
10040  IF TS THEN M$ = VT$(TP,C2). RETURN
10050 VT$(TP,C2) = M$. RETURN
15000 P1 = C0.P2 = CT:M = C0. HOME :SP = C0.SM = 10.TM = 50.FP1 = C0. GOSUB
     18150.LL$ = "0000"
15005 P2 = P2 + C1
15010  IF P2 > BC THEN  GOSUB 18200.P2 = C1
15020 LT$ = TB$(P2).LN$ =  LEFT$ (LT$,C4).LT$ =  MID$ (LT$,13). GOSUB 8000

15035  IF LN$ < LL$ THEN 15005
15040  IF PH$ = "SY," OR PH$ = "SN," THEN 16300
15050  IF PH$ = "T," THEN 17400
15060  IF PH$ = "H," THEN 16500
15070  IF PH$ = "M," THEN 16700
15080  IF PH$ = "R," THEN 15005
15085  IF PH$ = "P," THEN  INPUT "PRESS RETURN";PH$. GOTO 15005
15090  IF PH$ = "J," THEN 17290
15095  IF PH$ = "B," THEN  HOME : GOTO 15005
15100  IF PH$ = "U,  THEN 17260
15105  IF PH$ = "I," THEN 16100
15110  IF PH$ = "C," THEN 16400
15115  IF PH$ = "SY," OR PH$ = "SN," THEN 16300
15120  IF PH$ = "ES," THEN 17100
15125  IF PH$ = "G," THEN 17500
15130  IF PH$ = "E," THEN  PRINT . PRINT "DONE": END
15135  IF PH$ = "D," THEN 17600
15140  IF M THEN 15220
15150  IF PH$ = "TN," THEN 17400
15160  IF PH$ = "AN," THEN 16900
15170  IF PH$ = "MN," THEN 16700
15180  IF PH$ = "JN," THEN 17290
15190  IF PH$ = "UN," THEN 17260
15200  IF PH$ = "CN," THEN 16400
15205  IF PH$ = "IN,  THEN 16100
[illegible]  IF PH$ = "GN," THEN 17500
```