PILOT

REFERENCE MANUAL

1973

L. Turner
Department of Physics
Pacific Union College

TABLE OF CONTENTS

INTRODUCTION

PILOT is an author language for Computer Assisted Instruction (CAI). It is an implementation of PILOT 73 for a Hewlett-Packard 2000E Time-shared BASIC system. PILOT 73 is a hybridization of several author languages used in the San Francisco Bay area. Early in 1973 a group organized by Stanford Research Institute met and agreed upon a standardization called PILOT 73. These antecedents of PILOT have been used extensively for writing dialog type CAI programs.

There are several advantages of such an author language. First, the syntax is relatively simple so that within a short time most anyone can master the language. For traditional languages this is not necessarily true. It takes several weeks for a person to become proficient in BASIC. In general, most teachers do not have time to overcome inexperience in programming to be able to write effective teaching programs. Even more importantly, the construction of the PILOT language is such as to facilitate "dialog" type programming with a minimum of effort. One is able to write effective programs quickly. As a contrast, the BASIC language is designed for scientific computation. It does not lend itself easily to writing dialog type programs especially if the student's answer is to be examined for a given response. While it is certainly possible to write CAI programs in BASIC, the effort is very time consuming.

To be useful, any author language must be written as an interactive language. PILOT is implemented in BASIC for HP 2000E Time-shared system. This permits it to be used simultaneously with BASIC on this system.

It is quite true that the capabilities of the language PILOT are much less than that of BASIC. Its only virtues are the ease with which the language may be learned and the ease with which dialog CAI programs may be written.

Two cautions should be noted. Both concern themselves with possible misuse of the computer. First, many CAI programs are indistinguishable from ordinary programmed texts. This is not an effective use of the computer. The computer has several advantages over regular programmed texts, and effective use of the PILOT language will use these advantages. Whereas most programmed texts have no branching, that is, if a student learns rapidly there is no provision to skip over more questions on the same topic, a computer can easily branch to any other part of the program. In addition a computer can analyze the student's response without revealing the correct answer (It is not possible to "peek" at the expected response.) and it can give hints that lead the student to the correct answer. Also, a computer can keep track of how well a student progresses through the material. A well-written program forces the student to interact, not just passively read through the material.

Second, the PILOT language certainly lends itself to the writing of tests. However, in most cases it is easier and cheaper to use ordinary multiple copy techniques. One should not use the computer merely because it is new and novel. It is important to keep in mind the cost for a given result.

CAI can be an effective tool for the teacher just like audiovisual material, but it cannot solve all problems, nor should it be expected to take the place of effective existing (possibly cheaper) techniques; rather, it should supplement them.

PILOT was implemented by Lawrence E. Turner, Jr. of the Department of Physics, Pacific Union College. Any comments, reports of problems, or suggestions should be directed toward him.

A PILOT PROGRAM

A PILOT program consists primarily of statements and questions typed by the computer. The student's responses may be analyzed and appropriate action is then taken by the computer dependent on exactly what the student has typed. Thus a dialog between the computer and the student may be established.

The syntax of PILOT is relatively easy to master. An example will suffice to introduce one to the PILOT language.

```
10   R:  THIS IS A PILOT PROGRAM
20   R:
30   T:HELLO!  I AM YOUR FRIENDLY COMPUTER
40   T:  WHAT IS YOUR NAME +
50   A:
60   T:WELL $NAME HOW DO YOU USUALLY USE A COMPUTER??
70   A:
80   T:
90   T:THAT SOUNDS PRETTY GOOD!!
100  E:
```

This program contains four out of the thirteen possible statement types in PILOT. There are two features immediately obvious. The first is the statement number associated with each statement. The second is that each statement contains a colon (:). The character to the left of the colon gives the statement type.

The first statement is a REMARK statement which is not executed, it only appears in the listing to give information to someone looking at the listing. Statement 30 is a TYPE statement. Everything to the right of the colon is printed on the terminal. Similarly, so is statement 40; however, the plus (+) as a last character prevents the

carriage return-line feed that would normally occur at the end of the line. Thus statement 50, an ANSWER statement prints a question mark (?) at the second position after the last "E" in statement 4Ø. After printing the question mark the computer waits for the user to type in something from the terminal followed by a carriage return (ret). Whatever is typed-in is stored in the string variable NAME and is printed out in statement 6Ø, where $NAME is replaced by whatever the student typed-in for the response to statement 50. The last statement is an END statement which terminates the program.

A typical execution of this program is as follows. For this example the student's responses will be typed in lower case. On a conventional terminal however, everything will be in upper case only.

```
HELLO!  I AM YOUR FRIENDLY COMPUTER.
  WHAT IS YOUR NAME ?johnny c.

WELL JOHNNY C. HOW DO YOU USUALLY USE A COMPUTER??
?to play games and mess around

THAT SOUNDS PRETTY GOOD!!
```

If this were all the PILOT language could do, there would be little value in PILOT programs. A main virtue of a computer is the ability to analyze a student's response and to make decisions on the basis of what the student has typed. In PILOT this is accomplished by the MATCH statement and the use of a condition. Again an example.

```
1Ø   T:WHAT IS YOUR SEX +
2Ø   A:
3Ø   M:FEMALE,GIRL,WOMAN,LADY
4Ø   TY:I HOPE YOU ARE NOT A MEMBER OF WOMEN'S LIB!
5Ø   TN:I HOPE YOU ARE NOT A CHAUVINIST PIG!
```

In this case the MATCH statement (number 3Ø) has four possible alternatives separated by a comma (,). To illustrate the function of the MATCH statement, also consider two different executions of this portion of a PILOT program.

```
WHAT IS YOUR SEX ?i am a woman
I HOPE YOU ARE NOT A MEMBER OF WOMEN'S LIB!
```

and

```
WHAT IS YOUR SEX ?male
I HOPE YOU ARE NOT A CHAUVINIST PIG!
```

In the first cast the MATCH was successful; that is, somewhere within the response one of the alternatives (WOMAN) in the MATCH statement was found. Since the MATCH was successful, the TYPE statement with a YES condition is executed, the TYPE statement with the NO condition is skipped. In the second example, the MATCH failed, nowhere in the response was one of the MATCH alternatives found. Thus the TYPE statement with the NO condition was executed.

These examples introduce one to the general form of a PILOT program, five of the thirteen possible PILOT statements, and the concept of a condition. With this basis one could begin to write simple PILOT programs. Indeed some very effective programs have been written using only a portion of these ideas. However, some very powerful extentions are contained in the other statements.

Very briefly some of the more important of these are: the JUMP statement allows one to branch to any labeled statement, again depending on a condition, the DEMAND statement (.D) provides for accumulating the results of more than one MATCH statement to check for several different

responses in a single answer. With the COMPUTE statement the programmer can define variables which have numeric values associated with them. These variables may be given values, manipulated, and ultimately used in conditions or even printed on the terminal.

THE ESSENTIALS OF PILOT

This section contains a detailed description of each of the elements that make up the PILOT language. Hopefully, this manual will serve both as an instruction guide and a reference manual for PILOT.

Items contained within "<" and ">" are elements in the PILOT language. Throughout this manual the number zero is slashed (Ø) and the letter O is left unslashed.

PILOT PROGRAM STATEMENT

general form:

```
<line number> <instruction>:<object>
<line number> <label> <instruction>:<object>
<line number> <instruction><condition>:<object>
<line number> <label> <instruction><condition>:<object>
```

examples:

```
1ØØ  T:HI
2ØØ  *NAME A:$JOHNHENRY
3ØØ  JY:*QUESZ
4ØØ  *Q17 TN:SORRY YOU LOSE!
45Ø  C:X=5*Y-Z
```

comments:

A program statement contains a maximum of 72 characters.

The label and the condition are optional.

Note that every statement must contain one colon, (:). With some statement types the object may be blank.

Statements without a condition are termed unconditional and are always executed. Statements with a condition are termed conditional and may or may not be executed depending on the condition and previous responses to the program.

The only place where blanks are required is to separate the label from the instruction.

Extra blanks in the portion of the PILOT statement to the left of the colon are deleted.

The end of the program is designated by some last statement or possibly an END statement.

LINE NUMBER

examples:

```
1Ø   TY:HI
2ØØ  *NOW J:*THEN
1Ø97  M:BLUE,GREEN
```

purpose:

Line numbers serve no purpose to the PILOT language except to sequence the program statements as they are typed in and allow editing of the PILOT statements. Thus the program statements may be entered in any order. They are arranged in order of increasing line numbers.

comments:

Line numbers are positive integers in the range 1 to 9999.

To correct a statement, simply retype it with the same line number. To delete a statement type its line number.

The line number is separated from the rest of the statement by one or more blanks.

LABEL

general form:

*<string>

examples:

```
1Ø   *NAME T:MY NAME IS HP2000E!
2Ø   *AGE3 T:NOW IS THE TIME.
3Ø   *37A JY:*AGE7
4Ø   *NNN R:THIS IS A SUBROUTINE 'NNN'
```

purpose:

The label is an optional part of a PILOT program statement. It is used as a reference for a JUMP statement or a USE statement (subroutine jump).

comments:

The label consists of an asterisk (*) followed by a string consisting of letters and digits only and is ended by one or more blanks. It can be any length.

INSTRUCTION

general form:

T	TYPE
A	ANSWER
R	REMARK
E	END
M	MATCH
J	JUMP
U	USE
C	COMPUTE

N	TYPE
Y	TYPE

.T	UNREFERENCED TYPE
.M	EXTENDED MATCH
.I	INTEGER FUNCTION
.D	DEMAND
.X	RANDOM NUMBER FUNCTION

purpose:

The instruction designates the kind of statement.

comments:

The eight statements of the standardized PILOT 73 are designated by a single letter. The extensions which are unique to this implementation are designated by two characters, a period followed by a single letter.

For convenience the conditions Y and N indicate a TYPE statement (T) with Y or N condition respectively.

VARIABLE

examples:

A
B
F
Z

purpose:

The variables are in reality numeric variables and are used to store a numeric value.

comments:

Any single letter (A-Z) may be used as a variable. Thus there are 26 possible numeric variables.

Each variable may take on any real value.

Variables may be used as objects to ANSWER statements if preceded by the number sign (#).

They may be used in conditions, COMPUTE statements, INTEGER function statements, RANDOM number function statements, and referenced in TYPE objects.

STRING VARIABLE

example:

NAME
AGE73
ANSWER39
REMEMBERTHEALAMO

purpose:

The purpose is to take on the value of a string input as a response to an ANSWER statement.

comments:

The STRING VARIABLE consists of a string of characters consisting of only letters and digits.

The maximum length is 50 characters.

Initially all string variables are set to null (no characters).

NUMERIC CONSTANT

examples:

7.3
5
2
3E+2
4.21E-17

comments:

Any integer or decimal number is allowed. In addition, powers of ten notation is given by the 'E' designation.

Any value acceptable to 2000E BASIC is acceptable to PILOT. The PILOT interpreter does no checking on sizes of numbers; overflow and underflow error messages are generated by the BASIC system. The approximate ranges of allowed numbers are -1E+36 to -1E-36, 0, +1E-36 to +1E+36.

CONDITION

general form:

Y
N
G
B
(<variable>)
(<variable><relational operator><variable>)
(<variable><relational operator><signed number>)

examples:

```
2Ø1  JY:*THNM
273  TN:YOU BLEW IT!
1Ø82 T(B#7.32):ARE YOU CERTAIN??
2ØØØ T(D<=F):WELL NOW!!
```

purpose:

The condition allows the program to take alternative actions depending on the results of the previous sections. Every statement may optionally contain a condition.

comments:

The condition is followed immediately by the colon (:) in the statement.

There are three types of conditions. The first is designated by 'Y' or 'N' and depends on the results of previous MATCH statements. Each MATCH statement either fails or is successful. If it fails, then a match flag is set to a "no" state and all statements thereafter with a 'Y' are skipped. If the MATCH is successful, then the flag is set to a "yes" state and all statements thereafter with a 'N' are skipped.

The flag is initially set to a "yes" state and is reset to this state by an ANSWER statement. The match flag may be changed to either "yes" or "no" by the MATCH and DEMAND statements.

The second type is designated by a 'G' or 'B' and is used to test if a numeric response was contained in a previous ANSWER statement with a numeric variable as an object. If a valid number was present, then all statements with a 'B' condition are skipped. If no valid number was present, the 'G' condition statements are skipped. The numeric flag is initially set to "good" and is modified only by an ANSWER statement with a numeric variable as an object.

The third type is designated by a relation between two values enclosed in parenthesis. The execution of this statement depends only on the truth or falsity of the relation, not on the condition of the match flag or the numeric flag. The first value must be a variable. The second may be either a variable or a signed real numeric constant. The relational operators may be one of the following:

<	less than
<=	less than or equal to
=	equal to
#	not equal to
>=	greater than or equal to
>	greater than

It is also possible to test on the "truth" of a single variable. In this case the statement is skipped if the value is zero, "false". It is executed if the value is nonzero, "true".

CORE PILOT 73

This section describes the eight statements that comprise the standardized set of PILOT 73. These statements are represented by one-letter instructions.

There are two features of the basic PILOT 73 not implemented. The first is the continuation (indicated in PILOT 73 by ":" only). Of all the PILOT statements the only ones for which a continuation feature is applicable are: TYPE, COMPUTE, and MATCH statements. An effective continuation of the TYPE statement may be implemented by the use of '+', the concatenation operator. MATCH statements continuation may be effected by using a 'N' condition on the following MATCH statements. The inclusion of an explicit continuation feature grossly complicated the syntax analysis in an interactive environment. (See section 7.)

This implementation also requires an instruction for every labeled statement. Thus a statement consisting only of a label is not allowed. In practice one can always use a labeled REMARK statement if this construct is desired.

In addition, the object of the COMPUTE statement, while it is a correct BASIC statement, is restricted to BASIC assignment statements (without the preceding LET). These are further restricted to expressions involving only the arithmetic operators, i.e. no functions, no logical expressions, no Boolean connectives, and no MIN or MAX operators.

REMARK STATEMENT
R

general form:

```
<line number> R:<object>
<line number> <label> R:<object>
<line number> R<condition>:<object>
<line number> <label> R<condition>:<object>
```

examples:

```
1Ø   R:THIS IS A REMARK
2Ø   *REM R:I CAN'T BELIEVE IT!!
75   R:
1ØØ  R:THIS PROGRAM WAS WRITTEN BY JACK.
```

comments:

A REMARK statement is non-executable, that is, it does not result in anything being done! Thus it is meaningless to attach a condition.

A REMARK statement may be labeled and this label may be referenced in a JUMP or USE statement.

The object of the REMARK statement is used to convey information to someone who reads the listing of the program. It may be omitted.

TYPE STATEMENT
T

general form:

```
<line number> T:<object>
<line number> <label> T:<object>
<line number> T<condition>:<object>
<line number> <label> T<condition>:<object>
<line number> Y:<object>
<line number> N:<object>
<line number> <label> Y:<object>
<line number> <label> N:<object>
```

examples:

```
1ØØ  T:HI!
2ØØ  *NEW T:
3ØØ  Y:WELL $NAME YOU ANSWERED #N RIGHT!!
4ØØØ *OLD T(R=N):VERY GOOD!!
```

purpose:

The TYPE statement produces an output on the terminal.

comments:

The value of a numeric variable may be printed out by referencing the variable in the TYPE object. This is done by preceding it with a number sign (#) and following it with a plus (+), a blank, or the end of the line. The plus is not printed, and it deletes one blank following the printing of the value.

The value of a string variable may be printed by preceding it with a dollar sign ($) and following it with a plus, blank, or the end of the line. The plus is not printed and causes the next character following to be printed adjacent to the last character in the string variable.

If the string variable has not yet been given a value by an ANSWER statement, no characters are printed. If the string variable does not appear as an object to an ANSWER statement the variable name (with $ and ending character) is printed.

A plus (+) at the last character of the object is used for concatenation. The plus sign is not printed and the normal carriage return-line feed is not executed. Hence, the next printed character immediately follows on the same line.

The ending plus sign is stripped off before the line is printed. If a given word will not fit on a line it is printed on the next line. All breaks are between words just preceding the next non-blank character.

ANSWER STATEMENT
A

general form:

<line number> A:<object>
<line number> <label> A:<object>
<line number> A<condition>:<object>
<line number> <label> A<condition>:<object>

examples:

```
7ØØØ A:
8ØØØ *NAM A:$JOHNHANCOCK
8Ø1Ø AY:#Z
8Ø2Ø *MAJOR A:$MAJORDEP
```

purpose:

The ANSWER statement requests a response from the terminal.

comments:

The object may be null (no characters) or may consist of a dollar sign ($) followed by a string variable or a number sign (#) followed by a numeric variable (single letter).

All multiple blanks in the response are reduced to a single blank and the entire response is stored in the string variable if present. The first valid numeric quantity is stored in the numeric variable if it is present, and the numeric flag is set to "good". If no valid number is present in the response and a numeric variable is present, then the value of the numeric flag is set to "bad".

The ANSWER statement resets the match flag to a "yes" condition and the match counter to zero. (See DEMAND statement.)

The response of the single character "@" terminates the execution of the PILOT language program.

BASIC automatically supplies a question mark (?) when an ANSWER statement is executed and waits for a response.

The response is available for succeeding MATCH statements until another ANSWER statement is executed, even though the response is not stored in a string variable.

A single ANSWER statement may contain either a string variable or a numeric variable as an object but not both.

MATCH STATEMENT
M

general form:

<line number> M:<object>
<line number> <label> M:<object>
<line number> M<condition>:<object>
<line number> <label> M<condition>:<object>

examples:

```
1ØØ  M:WASHINGTON
2ØØ  *PRES M:WASHINGTON
3ØØ  MN:MARTHA,MRS. WASHINGTON
```

purpose:

The MATCH statement analyzes the user's response and sets the match flag according to whether the match failed or was successful.

comments:

The object for the MATCH consists of one or more alternatives separated by the character ",". If somewhere within the last ANSWER response one of these alternatives is found, the matching is halted and the match flag is set to "yes"; otherwise, if none of the alternatives is found, the match flag is set to "no". Blanks are significant within an alternative. Multiple blanks in the response are replaced by single blanks.

If the match was successful the match counter is incremented by one.

Previous responses may be used as MATCH patterns by enclosing the string variable (preceded by $) in commas just like any other alternative.

JUMP STATEMENT
J

general form:

<line number> J:<label>
<line number> <label> J:<label>
<line number> J<condition>:<label>
<line number> <label> J<condition>:<label>

examples:

```
1Ø1Ø J:*NEWQ
2Ø71 *NOW J:*THEN
2Ø77 JY:*AGE7
2Ø78 JN:*AGE8
4721 *ABC J(A<B):*C
```

purpose:

The JUMP statement provides for branching to any other labeled statement in a program.

comments:

The condition specifies whether the branch is executed or not.

During the setup phase for execution a check is made for unreferenced jumps. That is, a JUMP (or USE) statement object that is not a label that is present in the program.

No check is made for more than one statement with the same label. In that case the JUMP goes to the first one.

USE STATEMENT
U

general form:

<line number> U:<label>
<line number> <label> U:<label>
<line number> U<condition>:<label>
<line number> <label> U<condition>:<label>

examples:

```
2Ø   U:*SUBROUTINES
3Ø   U(X<7):*GOODIES
```

purpose:

The USE statement provides for a subroutine call to another portion of the program.

comments:

A USE statement generates a JUMP but with one important difference, the interpreter "remembers" from whence it is called. Upon execution of an END statement it returns to the next statement after the USE statement.

Subroutines may be nested up to 6 deep.

The main program is termed level zero, and each subroutine call increases the level number by one. A return from a subroutine (see END statement) decreases the level number by one.

Subroutines should not be used for small patches in a PILOT program since they involve at least two jumps which may take several seconds to execute.

END STATEMENT
E

general form:

```
<line number> E:<object>
<line number> <label> E:<object>
<line number> E<condition>:<object>
<line number> <label> E<condition>:<object>
```

examples:

```
9999 E:
1ØØ7 E:THIS IS THE END!!
8312 *ENDSTAT E:FINALLY
```

purpose:

The execution of an END statement results in a return from a subroutine (see USE statement) or a halting of the execution of the PILOT program.

comments:

The END statement may be conditional. If it is used as a return from a subroutine (see USE statement) control is transferred to the next statement following the USE statement. If the level is zero, i.e. the main program, execution of the PILOT program is terminated and control is passed to the interpreter.

The object of the END statement is used to convey information to someone who reads the listing of the program (just like the object of a REMARK statement). It may be omitted.

COMPUTE STATEMENT
C

general form:

```
<line number> C:<variable>=<expression>
<line number> <label> C:<variable>=<expression>
<line number> C<condition>:<variable>=<expression>
<line number> <label> C<condition>:<variable>=<expression>
```

examples:

```
1ØØ  C:N=17
127  C:R=X+Y↑3
4ØØ  C:X=(X-13)*Y/(-1+M)
41Ø  C:X=Y=0
```

purpose:

The COMPUTE statement is used to evaluate an expression and store the results in a numeric variable.

comments:

The expression is a combination of numeric variables, numeric constants and arithmetic operators which evaluates to a unique numeric value.

The arithmetic operators are:

+	addition
-	subtraction (or unary minus)
/	division
*	multiplication
↑	raise to a power

Evaluation is in order of priority as given by:

first	↑
second	/ or *
third	+ or -

Within a priority group evaluation is left to right within the expression.

Parentheses may be used at will to change the order of evaluation, with quantities in parentheses being computed first.

The expression is evaluated using the values of the variables before the expression is executed. This value is then stored in the variables appearing to the left of the equals (=). Note that this is a replacement statement not an algebraic equation.

Multiple replacement is allowed.

The syntax rules may be summarized as:

1. Only a single variable may appear to the left of an equals sign, (or possibly pairs formed by a single variable immediately followed by an equals sign).

2. A variable may be followed only by:
    a. an arithmetic operator
    b. a closed parenthesis
    c. end of the statement

3. A numeric quantity (without sign) may be followed only by:
    a. an arithmetic operator
    b. a closed parenthesis
    c. end of the statement

4. An open parenthesis may be followed only by:
    a. an open parenthesis
    b. a variable
    c. a numeric quantity
    d. a minus sign
    e. a plus sign (meaningless and is deleted)

5. A closed parenthesis may be followed only by:
    a. a closed parenthesis
    b. an arithmetic operator
    c. end of the statement

6. An arithmetic operator may be followed only by:
    a. an open parenthesis
    b. a variable
    c. a numeric quantity

7. An equals sign may be followed only by:
    a. an open parenthesis
    b. a variable
    c. a numeric quantity
    d. a minus sign
    e. a plus sign (meaningless and is deleted)

8. During any point in a left to right scan the number of closed parentheses may not exceed the number of open parentheses.

9. The total number of closed parentheses must equal the total number of open parentheses.

10. Blanks are meaningless and are deleted.

EXTENSIONS TO PILOT 73

Several extensions to PILOT 73 standard are possible. These increase the power and convenience of the language. They are designated by a two character instruction. For ease in syntax analysis the first is a period, and the second is a letter.

DEMAND STATEMENT
.D

general form:

```
<line number> .D:<integer>
<line number> <label> .D:<integer>
<line number> .D<condition>:<integer>
<line number> <label> .D<condition>:<integer>
```

examples:

```
1ØØ  .D:3
1ØØØ *DEM .D:2
2ØØØ .DY:4
```

purpose:

The DEMAND statement is used to accumulate the results of more than one MATCH. The match flag is set to "yes", if the number of successful matches since the last ANSWER is greater than or equal to the digit given in the object of the DEMAND statement. If this is not the case, the flag will be set to "no".

comments:

If the number of executed MATCH statements since the last ANSWER is less than the object of the DEMAND, the match flag will be set to "no".

If the object is zero, the match flag is always set to "yes".

The largest integer allowed for a DEMAND object is 99.

UNREFERENCED TYPE STATEMENT
.T

general form:

```
<line number> .T:<object>
<line number> <label> .T:<object>
<line number> .T<condition>:<object>
<line number> <label> .T<condition>:<object>
```

examples:

```
1ØØ  .T:HI!
2ØØ  *NEW .T:
3ØØ  .TY:THIS DOES NOT REFERENCE A VARIABLE #A OR $NAM +
4ØØØ *OLD .T(R=N):VERY GOOD!!
```

purpose:

The UNREFERENCED TYPE statement produces an output on the terminal, but does not recognize any numeric or string variable references.

comments:

The concatentation as designated by a plus (+) as the last character is still executed.

EXTENDED MATCH STATEMENT
.M

general form:

```
<line number> .M:<object>
<line number> <label> .M:<object>
<line number> .M<condition>:<object>
<line number> <label> .M<condition>:<object>
```

examples:

```
1ØØ  .M:[WASHINGTON],GEORGE
2ØØ  *PRES .M:[WASHINGTON,G.W.
3ØØ  .MN:MARTHA,MRS*WASHINGTON
```

purpose:

The EXTENDED MATCH statement analyzes the user's response and sets the match flag according to whether the match failed or was successful.

comments:

The object for the MATCH consists of one or more alternatives separated by the character ",". If somewhere within the last ANSWER response one of these alternatives is found, the matching is halted and the match flag is set to "yes". Blanks are significant. Multiple blanks in the response are replaced by single blanks.

A very powerful match pattern may be made using the character "*" somewhere within one of the alternatives. This will match any number of characters, including zero characters. In example 3ØØ above, for the second alternative, first "MRS" will be used to attempt a match, if successful then "WASHINGTON" will be used to attempt a match starting at the point where "MRS" was found.

Consider the sample program below

```
10   T:WHO IS THE PRESIDENT +
20   A:
30   .M:R*M*N*
```

The MATCH would be successful for the following responses:

```
RICHARD MILHOUS NIXON
RICHARD M. NIXON
R.M.N.
RMN
I REALLY AM NOT SURE
GEORGE MCGOVERN
HENRY 'THE MAN' KISSINGER
THAT IS A REAL MEAN QUESTION
ROOT MEAN SQUARE
RIGHT ON!  MY NUTTY COMPUTER
```

Thus this construct must be used with care! A second powerful pattern may be formed by using either a left bracket ([) to begin the pattern and/or a right bracket (]) to end it. A match is attempted on the pattern within the brackets and if a match is found a further check is made to see if an alphabetic character procedes (for [at beginning) or follows (for] at end). If it is found the match fails. Thus the brackets can force the match to look for words.

Consider the program above but with the use of brackets:

```
10   T:WHO IS THE PRESIDENT +
20   A:
30   .M[R*[M*[N
```

This would be successful for

```
RICHARD MILHOUS NIXON
RICHARD M. NIXON
R.M.N.
RIGHT ON!  MY NUTTY COMPUTER
```

As noted in the example brackets may be used with asterisks.

INTEGER FUNCTION
.I

general form:

```
<line number> .I:<variable>
<line number> <label> .I:<variable>
<line number> .I<condition>:<variable>
<line number> <label> .I<condition>:<variable>
```

example:

```
7ØØ  .I:H
71Ø  .I(M>10):M
```

purpose:

The purpose is to replace the current value of the variable with the greatest integer less than or equal to it.

comments:

This is a possibly useful function which cannot be easily synthesized with COMPUTE statements as given in this implementation.

RANDOM NUMBER FUNCTION
.X

general form:

```
<line number> .X:<variable>
<line number> <label> .X:<variable>
<line number> .X<condition>:<variable>
<line number> <label> .X<condition>:<variable>
```

examples:

```
27Ø  .X:X
28Ø  .X(R):Z
```

purpose:

The purpose is to provide a random number.

comment:

The value of the variable returned is a random number in the range zero to one.

HOW TO ENTER A PILOT PROGRAM

The PILOT interpretive system consists of two BASIC programs. The first is entitled 'PILOTS' and is used to enter the PILOT program and provides editing and syntax checking. The second is entitled 'PILOT' and is used to execute the PILOT program.

The PILOT program is stored on one to three disk files, but three program files must be declared during the syntax phase. The execution of 'PILOTS' produces an input loop. At each question either a PILOT command or a PILOT statement is required. Error messages indicate an incorrect input. The execution of 'PILOTS' may be halted at each input with the command STOP or with a 'ctrl C'. The 'break' should be used only during a LIST.

The three program files, in which the PILOT program is stored, are designated by the user. They should contain at least two records each. In practice the entire PILOT program may fit into one or two files. The PILOT program is stored three, four, five, or six statements per record. Thus a maximum length of 864 PILOT statements is allowed.

These files need not be the same size. A fourth scratch file of at least two records must be opened before entering a PILOT program. It may be longer but only the first two records are used in editing the three program files. A files statement must be declared in line 1 of 'PILOT' before executing.

Consider the following example:

```
OPEN-FILE1,30
OPEN-FILE2,30
OPEN-FILE3,20
OPEN-SCRACH,2

GET-$PILOTS
1 FILES FILE1,FILE2,FILE3,SCRACH
```

Since a total of 80 records have been opened the PILOT program must be less than 480 statements. The fourth file, 'SCRACH' must be at least 2 records.

Tape dumped by the LIS command may be read back if the terminal has an automatic reader control. Place the tape in the reader and turn it to start after a question mark has been printed by 'PILOTS' for a new statement. It will proceed to read in the tape. At the end of the tape an extra ? will have been read in, typing a carriage return will clear this.

PILOT COMMAND: LIST

general form:

LIS
LIS-<statement number>
LIS-<statement number>,<statement number>

examples:

LIS-20,400
LIST-3000
LIST

purpose:

The LIST command is used to list all or only a portion of the PILOT program on the terminal.

comments:

The first parameter is the starting statement number, the second is the ending statement number. The default is 1 and 9999 respectively.

Only the first three characters are required. The two paramenters are optional. If used, the first must be separated from the command by a hyphen (-), the second, if used, from the first by a comma.

The LIST command may also be used to dump a PILOT program on paper tape. This may be done by first running off leader (either 'here is' or CTRL SHIFT P), typing LIS (ret), then before the terminal begins to type out the program, turn on the tape punch.

PILOT COMMAND: NUMBER

general form:

NUM
NUM-<beginning number>
NUM-<beginning number>,<increment>

example:

NUM-1ØØ,1
NUM-1ØØØ
NUMBER

purpose:

The NUMBER command is used to renumber the line numbers of an existing PILOT program.

comments:

The two parameters are optional. If used, the first must be separated from the command by a hyphen (-), the second, if used, from the first by a comma.

The default quantities are 10 for beginning number and 10 for the increment.

If the last statement of the renumbered program is greater than 9999, the message "NO NUM" is printed and the program is not renumbered.

Only the first three letters of the command are required.

PILOT COMMAND: PURGE

general form:

PUR

purpose:

The PURGE command completely erases the PILOT program; that is, the program files are filled with eof's.

comments:

Only the first three characters are required.

PILOT COMMAND: SIZE

general form:

SIZ

purpose:

The SIZE command gives the number of statements, the number of records, and the number of the last statement.

comments:

Only the first three characters are required.

PILOT COMMAND: STOP

general form:

STO

purpose:

The STOP command is used to achieve an orderly exit from the program 'PILOTS'. The files are left in such a way as to allow future modification or execution.

comments:

Only the first three characters are required.

HOW TO RUN A PILOT PROGRAM

The BASIC program to execute a PILOT program is entitled 'PILOT'. It requires, in addition to the PILOT program files, a scratch file as long as necessary. These are to be entered in statement 1.

Consider the following example. A PILOT program has been stored on the files: FILE1 and FILE2.

```
OPEN-TEMP,48
GET-$PILOT
1 FILES FILE1,FILE2,TEMP
RUN
PILOT

NUMBER PROGRAM FILES?2
.
.
.
```

The first part of the execution of 'PILOT' sets up reference tables, checks for unreferenced JUMP statement and USE statements, and sets up a response table. The reference table consists of statement numbers and label names for all the labeled PILOT statements. The response table consists of string variables and a place to store a response for all ANSWER statements with a string variable as an object. Both these tables are stored on the scratch file.

The response table will take one-half of a record for each ANSWER statement with a string variable object. Thus in writing a PILOT program it is best to keep labels short and the number of ANSWER statement with string variable objects to a minimum. As a rough example; out of a PILOT program of 600 statements, one may

have perhaps 80 labeled statements. If these are relatively short, say less than 20 characters, then the reference table would take about 10 records leaving room for 76 stored responses in the next 38 records. In many cases it is not necessary to actual store the student's response in order to analyze it. These numbers are based on the assumption that the scratch file is the maximum length allowed of 48 records.

After the initial "set up" phase, the program 'PILOT' proceeds to interpret the PILOT language program. When the PILOT program is finished, control returns to 'PILOT' and the user is asked if he wishes to repeat the execution of the PILOT program. If the response is 'YES' (or just 'Y'), the PILOT program is re-executed, if not, control is returned to the BASIC system.

PILOT IN THE 2000E ENVIRONMENT

The HP 2000E BASIC has certain limitations that make implementing PILOT quite interesting and challenging. For this implementation it was decided to make the language as interactive as possible, to store the source code on files and interpret it directly, and to allow as much flexibility in the choice of file names as possible.

In order to allow the user any choice of file names, one cannot use the chain feature. Thus, as much as possible must be done in realtively few programs. Fortunately, there are two rather distinct phases: entering the program and checking for syntax errors and executing it. For many applications most users will execute pre-written programs. Thus there are two programs 'PILOTS' and 'PILOT' which implement the PILOT language. This does lead to a bit of clumsiness in going from the entry stage to the execution phase as might occur during "debugging".

It was also decided to do the syntax checking interactively as each statement was typed in rather than all at once when the program was executed. This virtually eliminates the possibility of using a continuation feature as standard PILOT 73 requires. Also the concern in this implementation is with number of lines rather than number of statements as in other implementations where causing many lines to be associated with one statement via continuation may be advantageous.

In the beginning it was decided to store only three statements per record. This speeds up the syntax phase and greatly simplifies deletion, insertion, or replacement of statements. It is somewhat wasteful of file space, especially if large numbers of REMARK and TYPE statements are used with a blank object.

The limit of 432 statements (or lines) that this would imply did not seem to be a great restriction on the size of a PILOT program. In practice most PILOT programs are much shorter. However, after examining several PILOT programs it was discovered that only about 45 percent of the available space in the files was being used. By packing statements as much as possible approximately 90 percent was used. Currently PILOT puts up to six statements per record, this results in about 85 percent utilization of the possible space in increases the size of the maximum PILOT program to 864 statements. In practice an upper limit of 750-800 statements can be expected.

SAMPLE PROGRAM

The following sample program is included to provide an example of a PILOT program. It is written to include all the possible features in PILOT. It is not necessarily an effective or even a useful CAI program.

```
1000 R:     THIS IS A DEMONSTRATION PILOT PROGRAM
1010 R:
1020 R:   PACIFIC UNION COLLEGE, L. TURNER, MAY 73
1030 R:
1040 T:   HELLO!  I AM YOUR FRIENDLY HP 2100A.
1050 T:
1060 T:WHAT IS YOUR NAME +
1070 A:$NAM
1080 T:
1090 T:THAT IS A NICE NAME $NAM+.  I RATHER LIKE IT!!
1100 T:
1110 R:   GET SEX, CHECK FOR 'FEMALE' FIRST SINCE IT CONTAINS 'MALE'
1120 T:  IF YOU DON'T THINK IT TOO PERSONAL . . .
1130 *SEX T:WHAT IS YOUR SEX +
1140 A:$SEX
1150 T:
1160 M:FEMALE,GIRL,WOMAN,LADY,FEMIN,GAL
1170 Y:  I THOUGHT SO.  YOU'RE CUTE.
1180 CY:M=1
1190 JY:*AGE
1200 M:MALE,MAN,BOY,MASCUL,GUY
1210 Y:  THAT WAS WHAT I WAS AFRAID OF!
1220 CY:M=0
1230 JY:*AGE
1240 T:ARE YOU TRYING TO FOOL ME??
1250 T:IT'S NOT NICE TO FOOL YOUR FRIENDLY HP COMPUTER!!
1260 J:*SEX
1270 *AGE T:I WOULD ALSO LIKE YOUR AGE PLEASE +
1280 A:#A
1290 T:
1300 TB:TSK, TSK, TSK!
1310 JB:*AGE
1320 T(A<8):I DON'T BELIEVE YOU!!!
1330 T(A>75):ARE YOU REALLY THAT OLD???
1340 J(A<5):*AGE
1350 .I:A
1360 R:
1370 R:   N = NUMBER OF QUESTIONS
1380 R:   R = NUMBER OF CORRECT INITIAL RESPONSES
1390 R:   F = NUMBER OF TIMES FOR EACH QUESTION
1400 R:
1410 C:R=N=0
1420 T:  I HAVE A FEW QUESTIONS FOR YOU. +
1430 *BEG T:ARE YOU READY+
1440 A:
1450 M:Y,OK,O.K,RIG,REA,GO
1460 N:  NOW WHAT'S THE MATTER??
1470 JN:*BEG
1480 R:
1490 C:N=N+1
1500 C:F=1
```

```
151Ø *Q1 T:
152Ø T:WHO IS THE PRESIDENT OF THE U.S. +
153Ø A:
154Ø M:TRICK,HENRY,KIS,SPIR,AGN
155Ø Y:I SUSPECT YOU ARE A DEMOCRAT!!
156Ø M:NIXON
157Ø Y:MY, YOU ARE UP TO DATE!!
158Ø JY:*Q2
159Ø M:DICK,RICH,R.
16ØØ Y:YOU KNOW HIS FIRST NAME, HOW ABOUT THE LAST NAME??
161Ø N:YOU MUST BE A DEMOCRAT OR ARE YOU AFRAID THIS TERMINAL IS BUGGED?
162Ø C:F=F+1
163Ø J:*Q1
164Ø R:
165Ø *Q2 T:
166Ø C(F=1):R=R+1
167Ø T(M=1):FOR A CUTE CHICK +
168Ø T(M=0):FOR A HANDSOME GUY +
169Ø T:OF #A+YEARS, YOU TOOK ONLY #F+
17ØØ T(F=1):TRY!!
171Ø T(F>1):TRIES!!
172Ø C:F=1
173Ø C:N=N+1
174Ø R:
175Ø T:
176Ø *QUES2A T:CAN YOU NAME AT LEAST THREE COMPUTER MANUFACTURERS??
177Ø A:
178Ø M:YES,THINK SO,WILL TRY,RIGHT,OF COURSE,YEA
179Ø Y:GOOD, WHAT ARE THEY??
18ØØ JY:*QUEST2B
181Ø M:NO,CAN'T,CANNOT,DON'T
182Ø Y:WELL, TRY ANYWAY!
183Ø *QUEST2B AY:
184Ø T:
185Ø .M:IBM,INT*BUS*MAC
186Ø .M:HP,H-P,HEW*PAC
187Ø .M:DEC,PDP,DIG*EQ
188Ø .M:NOVA,DAT*GEN
189Ø M:XDS,XEROX
19ØØ .M:GE,GEN*EL
191Ø .M:CDC,CON*DAT
192Ø M:BURRO,BOUR
193Ø .M:HON*WELL
194Ø M:UNIVAC
195Ø .M:RCA,RAD*CORP*AM
196Ø .M:NCR,NAT*CA*REG
197Ø M:VARIAN
198Ø .D:4
199Ø Y:VERY, VERY GOOD!!  YOU GOT MORE THAN THREE, +
2ØØØ JY:*QUESBB
```

```
201Ø .D:3
202Ø Y:RIGHT ON, +
203Ø JY:*QUESBB
204Ø .D:2
205Ø Y:ALMOST, YOU GOT TWO CORRECT!
206Ø C:F=F+1
207Ø JY:*QUES3
208Ø .D:1
209Ø JN:*NOWAY
21ØØ .M:IBM,INT*BUS*MAC
211Ø Y:SO YOU ARE ONE OF THOSE HEADLESS CHICKENS WHO THINK THAT +
212Ø Y:IBM IS THE WORLD'S ONLY COMPUTER MANUFACTURER!
213Ø Y:
214Ø *QU2B T:I THINK YOU SHOULD TRY AGAIN!
215Ø C:F=F+1
216Ø .D:Ø
217Ø J:*QUEST2B
218Ø *NOWAY T:NO WAY!  YOU DIDN'T EVEN GET ONE RIGHT.
219Ø J:*QU2B
22ØØ *QUESBB T:$NAM+!  I CAN TELL YOU'RE AN EXPERT!
221Ø *QUES3 R:
222Ø C(F=1):R=R+1
223Ø T:
224Ø T:YOU DID SO WELL ON THAT LAST QUESTION, LET ME TRY A MORE +
225Ø T:THOUGHT PROVOKING ONE!
226Ø T:
227Ø T:I AM THINKING OF A NUMBER BETWEEN 0 AND 100.  CAN YOU GUESS +
228Ø T:IT???
229Ø U:*GOGETRANDOMNUMBER
23ØØ *TRYAGAIN C:H=1
231Ø *QNUM T: #H+
232Ø A:#T
233Ø TB:COME ON, TRY A NUMBER!
234Ø JB:*QNUM
235Ø J(T=Y):*VERYGOOD
236Ø C:H=H+1
237Ø T(T>Y):TOO LARGE!
238Ø T(T<Y):TOO SMALL!
239Ø J:*QNUM
24ØØ R:
241Ø *VERY GOOD J(H>7):*OVER
242Ø T:THAT IS VERY GOOD $NAM+, YOU DID IT IN SEVEN TRIES OR LESS!
243Ø J:*ENDNUM
244Ø *OVER T:YOU WASTED A FEW GUESSES, THE MOST IT SHOULD TAKE IS +
245Ø T:SEVEN, BUT $NAM+, YOU TOOK #H+GUESSES!
246Ø *ENDNUM T(F=4):ONLY ONE MORE TIME.
247Ø J(F>=5):*NEXTQ
248Ø T:WOULD YOU LIKE TO TRY IT AGAIN +
249Ø A:
25ØØ M:YES,RI,PLE,OF COUR
```

```
251Ø JN:*NEXTQ
252Ø U:*GOGETRANDOMNUMBER
253Ø C:F=F+1
254Ø J:*TRYAGAIN
255Ø R:
256Ø *NEXTQ T:
257Ø C:N=N+1
258Ø T:I HAVE ONE LAST QUESTION FOR YOU $NAM+!  WHAT DOES CAI MEAN??
259Ø A:
26ØØ .M[COMP*[ASS*[INS,[COMP*[AID*[INST
261Ø Y:EXCELLENT $NAM+!   +
262Ø CY:R=R+1
263Ø N:YOU BLEW IT!   +
264Ø T:CAI STANDS FOR COMPUTER ASSISTED INSTRUCTION.  THAT'S WHAT  +
265Ø T:WE'RE DOING RIGHT NOW!!
266Ø T:
267Ø T:WELL, $NAM+, OUT OF THE  #N+QUESTIONS I ASKED (NOT COUNTING +
268Ø T:THE NUMBER GUESSING ONE), YOU GOT #R+CORRECT ON THE FIRST  +
269Ø T:TRY.  +
27ØØ T(N=R):THAT IS JUST ABOUT PERFECT!
271Ø T(R=0):NOT EVEN ONE RIGHT!  WHAT'S A COMPUTER TO DO?
272Ø T(R=1):YOU GOT AT LEAST ONE CORRECT.
273Ø E:  THIS IS THE END OF THE MAIN LINE PROGRAM
274Ø *GOGETRANDOMNUMBER R:
275Ø .X:Y
276Ø C:Y=Y*99
277Ø .I:Y
278Ø C:Y=Y+1
279Ø E:
```

PILOT SYNTAX

The PILOT syntax is described on the following pages. The characters "<" and ">" enclose an element of PILOT. The character "::=" is to be read: "is defined as". The character "|" is to be read as "or".

SYNTAX OF PILOT

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<integer> ::= <digit>|<integer><digit>

<decimal number> ::= <integer>|<integer>.|<integer>.<integer>|
.<integer>

<number> ::= <decimal number>|<decimal number><exponent>

<exponent> ::= E<integer>|E+<integer>|E-<integer>

<signed number> ::= <number>|+<number>|-<number>

<letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|
V|W|X|Y|Z

<letter digit> ::= <letter>|<digit>

<ld string> ::= <letter digit>|<ld string><letter digit>

<character> ::= *any ASCII character except: null, line feed, carriage return, X^c, x-off, ←, ", rubout.*

<m character> ::= *any ASCII character except: null, line feed, carriage return, X^c, x-off, ←, ", comma, rubout.*

<e character> ::= *any ASCII character except: null, line feed, carriage return, X^c, x-off, ←, ", comma, *, rubout.*

<null> ::= *no characters*

<string> ::= <character>|<null>|<string><character>

<m string> ::= <m character>|<null>|<m string><m character>

<e string> ::= <e character>|<null>|<e string><e character>

<variable> ::= <letter>

<string variable> ::= <ld string> *maximum length is 50*

<program statement> ::= <line number><PILOT statement> carriage return|
<line number><label>␣<PILOT statement>carriage return

<line number> ::= <integer> *allowed range is: 1 to 9999*

```
<label>                 ::= *<ld string>

<condition>             ::= <YN condition>|<GB condition>|<conditional>

<YN condition>          ::= Y|N

<GB condition>          ::= G|B

<conditional>           ::= (<variable>)|
                            (<variable><relational><primary>)

<relational>            ::= <|<=|=|#|>=|>

<primary>               ::= <variable>|<signed number>

<PILOT statement>       ::= <core statement>|<extended statement>

<core statement>        ::= <TYPE statement>|<ANSWER statement>|
                            <MATCH statement>|<REMARK statement>|
                            <JUMP statement>|<USE statement>|
                            <END statement>|<COMPUTE statement>

<extended statement>    ::= <EMATCH statement>|<ETYPE statement>|
                            <DEMAND statement>|<INTEGER statement>|
                            <RANDOM statement>

<TYPE statement>        ::= <TYPE instruction>:<TYPE object>|
                            <TYPE instruction>:<TYPE object>+

<TYPE instruction>      ::= T|T<condition>|<YN condition>

<TYPE object>           ::= <string>|
                            <TYPE object>$<string variable><endr><TYPE object>
                            <TYPE object>$<string variable>|
                            <TYPE object>#<variable><endr><TYPE object>|
                            <TYPE object>#<variable>

<endr>                  ::= ø|+

<ANSWER statement>      ::= <ANSWER instruction>:<ANSWER object>

<ANSWER instruction>    ::= A|A<condition>

<ANSWER object>         ::= <null>|$<string variable>|#<variable>

<MATCH statement>       ::= <MATCH instruction>:<MATCH object>

<MATCH instruction>     ::= M|M<condition>

<MATCH object>          ::= <alternative>|<MATCH object>,<alternative>

<alternative>           ::= <m string>|$<string variable>
```

```
<REMARK statement>      ::= <REMARK instruction>:<REMARK object>

<REMARK instruction>    ::= R|R<condition>

<REMARK object>         ::= <string>

<JUMP statement>        ::= <JUMP instruction>:<JUMP object>

<JUMP instruction>      ::= J|J<condition>

<JUMP object>           ::= <label>

<USE statement>         ::= <USE instruction>:<USE object>

<USE instruction>       ::= U|U<condition>

<USE object>            ::= <label>

<END statement>         ::= <END instruction>:<END object>

<END instruction>       ::= E|E<condition>

<END object>            ::= <string>

<COMPUTE statement>     ::= <COMPUTE instruction>:<COMPUTE object>

<COMPUTE instruction>   ::= C|C<condition>

<COMPUTE object>        ::= <destination><expression>

<destination>           ::= <variable>=|<destination><variable>=

<expression>            ::= <term>|<expression>+<term>|<expression>-<term>|
                            -<term>|+<term>

<term>                  ::= <factor>|<term>*<factor>|<term>/<factor>

<factor>                ::= <subfactor>|<factor>↑<subfactor>

<subfactor>             ::= <variable>|<number>|(<expression>)

<EMATCH statement>      ::= <EMATCH instruction>:<EMATCH object>

<EMATCH instruction>    ::= .M|.M<condition>

<EMATCH object>         ::= <pattern>|<EMATCH object>,<pattern>

<pattern>               ::= <complex>|<pattern>*<complex>

<complex>               ::= <e string>|<word>

<word>                  ::= [<e string>|[<e string>]|<e string>]
```

```
<ETYPE statement>       ::= <ETYPE instruction>:<ETYPE object>|
                            <ETYPE instruction>:<ETYPE object>+

<ETYPE instruction>     ::= .T|.T<condition>

<ETYPE object>          ::= <string>

<DEMAND statement>      ::= <DEMAND instruction>:<DEMAND object>

<DEMAND instruction>    ::= .D|.D<condition>

<DEMAND object>         ::= <integer>                    allowed range 0 → 99

<INTEGER statement>     ::= <INTEGER instruction>:<INTEGER object>

<INTEGER instruction>   ::= .I|.I<condition>

<INTEGER object>        ::= <variable>

<RANDOM statement>      ::= <RANDOM instruction>:<RANDOM object>

<RANDOM instruction>    ::= .X|.X<condition>

<RANDOM object>         ::= <variable>
```

ERROR MESSAGES

SYNTAX PHASE: PILOTS

MISSING OR PROTECTED FILE	'PILOTS' needs exactly four files
F # x TOO SMALL	file number x has only one record
????	entry has no statement number and is not a valid command
UNRECOGNIZED	entry has a valid statement number but a non-recognized instruction
LBL ERR	label error
NO COLON	missing or misplaced colon
CON ERR	condition error
OBJ ERR	object error
EXP ERR: xx	expression error occurring after the rightmost =, offending characters are printed
PAR ERR	parenthesis error due to number of close parentheses exceeding open parentheses at a given point, or unmatched open parentheses
FILES FULL	the number of statements already stored is such that the new entry may cause the last few statements to be lost
NO NUM	unable to renumber because last statement number would exceed 9999
BAD INPUT RETYPE	more than 72 characters were entered

SETUP PHASE: PILOT

SCR FILE OVRFLW	scratch file is too small for reference and response tables
UNREF, JMP xxxx	a JUMP or USE object to a label that does not exist, the statement number is given

EXECUTION: PILOT

USE ERR	subroutines nested more than six deep

Note: no special checking is made for numeric overflow. The BASIC system may generate several possible error messages not given above. In this case please check in 2000E: A GUIDE TO TIME-SHARING BASIC.