NEVADA

```
PPPPPP    IIIIIII    L          OOOOO     TTTTTTT
P    P       I       L         O     O       T
P    P       I       L         O     O       T
PPPPPP       I       L         O     O       T
P            I       L         O     O       T
P            I       L         O     O       T
P         IIIIIII    LLLLLLL    OOOOO        T
```

PROGRAMMER'S REFERENCE MANUAL
Fourth Edition 1982

John A. Starkweather
University of California, San Francisco

Published by Ellis Computing
3917 Noriega Street
San Francisco 94122
(415) 753-0186

NEVADA PILOT

TABLE OF CONTENTS

SECTION PAGE

SECTION 1

NEVADA PILOT: GENERAL INFORMATION

1.0 INTRODUCTION

PILOT is a programming language for interactive programs, i.e., for those programs in which the user participates in a kind of conversation with the computer by typing responses to questions posed by the computer. PILOT stands for Programmed Inquiry, Learning Or Teaching, and has been used most commonly as a language for computer-assisted instruction. It was first developed by John A. Starkweather at the University of California in San Francisco and has been implemented on a variety of large and small computers.

This guide provides a description and operating procedures for a version of PILOT that runs on an 8080, Z-80, or 8085 microcomputer equipped with diskette storage and a visual display terminal. The latter can be either a memory-mapped display or the more common terminal (CRT) with serial connection to the computer.

Portions of this manual and a related PILOT, Version 1.1, were first prepared for the National Library of Medicine, Bethesda, Maryland. Other portions were developed for PILOT Version 2.2, a cassette tape version for the Processor Technology Sol Computer.

1.1 WHAT CAN BE DONE WITH PILOT?

PILOT enables a person without any prior computer experience to develop and test dialogue programs for use in an instructional or training context. The simplicity of the instruction format, as well as the conversational structure of the system, make it easy for a teacher to prepare a program and easy for a student to use it, with a minimum of training.

PILOT makes it possible, for example, to present the student with a reading passage, allow time to study it, and then ask a series of questions to check on comprehension. The program might include computer responses keyed to the answer a student has given, or it might ask for the student's reaction to the passage, scan his response, and comment or provide suggestions based on that response.

A teacher can prepare a vocabulary quiz that keeps track of a student's score and then moves on or repeats a lesson, depending on the student's performance or choice. A PILOT program can introduce a mathematical word-problem and offer the solution, step by step, or it can give the student the opportunity to discover as many of the steps as possible, with the computer hinting along or even revealing a step, when necessary.

PILOT is also a vehicle for learning about computers. You can write a program that teaches how to write a PILOT program and let the student save the program, so that it can be executed or revised on another occasion. (Or the student can write, edit, and run the program in the same sitting.)

A user can have considerable control over the order of events in a program, and is therefore not likely to find the experience intimidating. Users can select what they want to read, or decide whether to repeat an exercise or execute another program. They can even determine where responses will appear on the screen or make them appear with reverse video (i.e., white letters on black, for a screen set to black letters on white, or vice-versa) to emphasize a particular word in the answer. Users can keep their answers, list them, and review the exercises with which they had difficulty. PILOT is also suitable for writing conversational games.

Section 3 of this manual introduces the elements of PILOT by proposing a simple program design and showing how the PILOT language can be used to realize and expand it.

Just as in a spoken language, you can express many ideas and accomplish many tasks by combining a handful of simple statement types.

1.2 WHY NOT BASIC, OR ANOTHER PROGRAMMING LANGUAGE?

It is possible to write interactive programs in a general purpose programming language such as BASIC, APL, FORTRAN, or PL/1 instead of in PILOT. The reverse is certainly not true, for PILOT is a specialized language oriented toward dialogues, drills, tests, etc., rather than toward some kinds of computation handled well by general purpose languages. The advantage of PILOT over general purpose languages such as BASIC is that interactive programs are easier to write in PILOT. If a BASIC program is to handle free-response dialogue, the programmer must often make unwieldy arrangements for processing input and comparing words or portions of words that the program must recognize. In PILOT this kind of processing is easy, and the programs are more readable by human beings than they might be if they were written in some other computer language.

1.3 TERMS AND CONVENTIONS

The following terms will appear frequently. The definitions given here are not technical, nor even very complete, but should provide some insight into the kinds of concepts that are being described when these words and expressions are used.

A BUFFER is a temporary storage space set aside within the computer memory for a collection of possible items. A PROGRAM BUFFER, for example, is space set aside for the information necessary to execute a certain program.

A CHARACTER may be a letter, a punctuation mark, a number, or a symbol. Note that CHARACTERS, even those which represent numbers, are not involved in computations, i.e., the symbol "3" plus the symbol "2" does not yield the number "5", the symbol "5", or, in fact, anything meaningful for our purposes. A CHARACTER is not a quantity, but a representation. A CONTROL CHARACTER is formed by hitting the control key and another key simultaneously. Some CONTROL CHARACTERS function as commands to the computer and are not displayed on the screen. Others are displayed as special graphic symbols.

CODE has two meanings. In its more standard usage, the CODE for something is a sign that is used to stand for the thing. A more specialized meaning of the word is: that sequence of instructions, in a computer language, that makes up a computer program, or part of a computer program. When you write a PILOT program, you will be writing CODE which is translatable into activity by the computer.

A COMMAND is a direct instruction to the computer. A STATEMENT, by contrast, is indirect. It is entered as a line in a program, and initiates no immediate activity. Almost every "sentence" in the PILOT language can be used both as a program STATEMENT and as a direct COMMAND. The convention we will adopt for this is to use the word STATEMENT, except in the context of immediate execution. (The COMMANDS which belong to the PILOT EDITOR are not part of the PILOT language, and always result in immediate execution.)

A CONSTANT is an item of data that has a fixed, or constant, value. 32 is a CONSTANT.

A CURSOR is the rectangle or blinking underline that is visible on the video display screen. It serves to show the screen position at which the next CHARACTER will be displayed.

A FILE is a collection of related information, usually named and referenced as a unit. Think of a FILE as a way of saving such a body of information for subsequent use. A FILE can contain a program, text, or data of various kinds.

EDITing a FILE is modifying that FILE in any way, including typing it into the computer for the first time. The EDITOR is that program, or that part of a program, which enables you to use certain direct COMMANDS to EDIT a FILE.

EXECUTION is the actual "running" of a program. When execution occurs, the computer follows programmed instructions.

A DATA COLLECTION FILE is a FILE set up to store information that the computer receives, usually from the keyboard. A FILE is very different from a BUFFER, in that 1) a BUFFER is a temporary storage area that exists only during the operation of a program, whereas a FILE (at least for our present purposes) is a more permanent record, 2) a FILE is usually recorded on some kind of device (e.g.,tape or disk) outside the "memory" of the computer, whereas a BUFFER does not exist, theoretically, when the computer is inactive, and 3) a FILE consists of saved information, whereas a BUFFER is a place where information can temporarily reside. Recording a FILE on an external device is called WRITING the FILE; retrieving a FILE from storage is called READING it.

An INTEGER is a positive or negative whole number. 5,-5, and 0 are all INTEGERS.

LOADING a program is reading into the computer the instruction set that makes up the program. You cannot execute a PILOT program that you have saved on disk until you have LOADED it. In some systems LOADING a program will also cause it to EXECUTE.

MEMORY is the storage area of the computer. One of the important facts to remember about computer MEMORY is that it is limited. It is possible for a program, or a DATA COLLECTION FILE, to be too large for the computer to accommodate. We will discuss ways of making sure that the programs we are using are not too big to fit into MEMORY. (A BUFFER is one way of allocating a portion of MEMORY.)

A STRING, or CHARACTER STRING, is a set of consecutive CHARACTERS. "DOG," "23," and "This is a sentence" are all STRINGS.

FORMAT is used in this manual to refer to the required form or structure of a PILOT statement. Each statement type is headed by a description of its format, using angle brackets to indicate variable content, and square brackets to indicate optional content. Thus, the T statement is partially described as follows:

```
[<label>] T [<cond>] : <message>
```

label, cond, and message are elements of variable content. label and cond are optional and may not be present.

A VARIABLE, by contrast to a CONSTANT, is a name to which different values can be assigned. The equation x = 3 causes a value of 3 to be assigned to a variable named x. Computer languages have rules for designating VARIABLES. We will discuss rules for PILOT variables in Section 3.4.

The following conventions will be used here:

1. PILOT statements and commands will be represented in capital letters, even though you do not really have to enter them in capital letters. EDIT, LOAD, and BYE are all examples of PILOT commands.

2. Where an expression is enclosed by angle brackets <like this>, the expression tells what kind of item should be typed as part of a statement. For example, <number> indicates that a number should be entered (without the brackets).
<cr> indicates that a carriage return should be struck.

3. Square brackets [like these] enclose an optional element in a statement. For example, [<filename>] indicates that the use of a filename is optional. The brackets, again, are not literal. That is, they are not part of the final statement.

4. In examples that contain PILOT code, two rows of dots are used to indicate code that was omitted because it is irrelevant to the example.

```
<statement>
  ....
  ....
<statement>
```

5. "cond" is an abbreviation for "conditional expression." A conditional expression is used in cases where a statement should be executed only if a specific condition is met. Conditional expressions are discussed at length in Section 3.2.

6. Regard the video display screen as divisible into horizontal ROWS and vertical COLUMNS. When you type in a command, the characters that you type will fall in the same ROW, but in consecutive COLUMNS. It is not necessary to begin typing a PILOT statement in any particular COLUMN within a ROW. In the examples in this manual, indentation may be used for the sake of readability; it has no effect, whatever, on the operation of the program.

7. Quotation marks (" ") are used to enclose a message that appears on the screen, e.g., "READY" or "*NO ROOM".

8. Control characters, those formed by depressing the control key in conjunction with another key on the keyboard, are denoted by ctrl-<character>; for example, ctrl-A.

SECTION 2

INSTALLATION AND OPERATION

2.1 Installing PILOT on your equipment

The PILOT package contains a programmer's reference manual and a disk. The disk contains the following files:

NVPILOT.COM
PLTST.PLT
PLTST2.PLT
TST-C.PLT
TST-S.PLT
WAPP.PLT
READ-ME.TXT
TST-MJ.PLT
TST-JM.PLT
TST-VN.PLT
TST-I.PLT
TST-WILD.PLT
HODGE.PLT
NVPILOT.PRN

Your first step should be to prepare a systems disk, a disk with CP/M (either Version 1.4 or 2.2), and copy these programs onto it. If you have two disk drives, you can prepare a CP/M disk containing PIP, place it in drive A, place the PILOT disk in drive B, and issue the following command: `A> PIP A:=B:*.*` Put the original PILOT disk away in a safe place.

Make sure that your default disk (the logged-on disk drive) has enough room for the 16K PILOT.COM file that will be created there. You may need to delete some other files before taking the next step.

Next, configure PILOT for your CRT or video display by issuing the command: `A> NVPILOT` The program will present you with a choice of terminals and prompt your next steps. PILOT.COM will be created on the default disk by this process. This is the PILOT system configured for your equipment.

If your terminal is listed, you need do no more than press one key in order to proceed. If your terminal is not listed, press "U" for unknown, and you will be asked a series of questions about its characteristics. See Section 8 for an example of the interaction involved in the configuring process.

During installation and after choosing your terminal, you will be asked to choose between full PILOT for author use and an abbreviated PILOT for student use. The student version does not give access to the EDIT function and avoids potentially damaging immediate commands. In addition, you will be asked to choose whether you wish the DIR command (that displays the disk directory) to show all files for just files of type .PLT .

After PILOT has been configured run the test program PLTST to make sure everything is working correctly: `A> PILOT PLTST` This program is printed in Section 7 so that you can also use it to learn some of the details of PILOT programming. If PILOT works properly you may remove NVPILOT.COM from your operational disk to free more space.

If your computer is an Apple, you must run the Microsoft program to configure the SoftCard: MBASIC CONFIGIO Select the Soroc IQ-120 as your software terminal. Also, using the CONFIGIO, delete the reasignment of all control keys. If you are using the Videx 80 column board on your Apple, the Control-A key has been reassigned to Control-S and the Control-S key has been reassigned to Control-D.

The file NVPILOT.PRN contains information for further configuring to meet special needs or to change keys used for special functions. For example, some models of Televideo terminals that do not respond to codes for insert line or delete line require that you use this method to obtain these functions. Select the Televideo choice when you run NVPILOT and if these functions do not work, then use DDT to place a zero in the IL and DL locations. This will force a rewriting of the screen for these functions.

2.2 Running PILOT

Once the system prompt from CP/M has been obtained ("A>"), you may start PILOT in one of two ways:

Type: PILOT (Return)

> This puts PILOT into operation in immediate mode with an empty program area. A title and copyright notice will appear, followed by a list of available immediate commands, and the word "READY".

or Type: PILOT <file name> (Return)

> where <file name> indicates the name of an existing PILOT program. PILOT first loads the referenced PILOT program text and begins execution of the program in run mode. The file name can be prefaced by a drive, as in: A> PILOT B:MYPROG

2.3 PILOT Operating Modes

At any time, PILOT is in one of three operating modes; either **immediate** mode, **run** mode, or **edit** mode. In immediate mode, a limited number of commands are available which will be executed as soon as a terminating carriage return is entered. In run mode, execution is under control of a previously prepared PILOT program, while in edit mode, the user is able to develop or change the text of a PILOT program.

Immediate Mode

PILOT starts in immediate mode. The system signals that it is ready to accept an immediate command by displaying the prompt "READY" and displaying a visible cursor (normally a blinking underline character) at the beginning of the following line. A display of available immediate commands may be seen by typing "?" (Return). These are described in Section 4.7. The system remains in immediate mode until you type either "RUN" or "EDIT" which causes a change to run mode or to edit mode.

Run Mode

In run mode PILOT interprets and executes (follows the instructions of) the program currently in the program buffer.

The program will continue to execute until:

> a PILOT End statement is executed in the program with no return pending from a Use statement,
>
> or there are no more statements to execute,
>
> or the user types "\" (Return)
> (ctrl-F may be substituted if "\" is not available).

When one of these conditions occurs PILOT returns to immediate mode.

In some of the program examples, immediate mode is referred to as "PILOT restart".

Edit Mode

In edit mode PILOT displays the contents of the program buffer and allows addition or changes to it. Edit mode provides a variety of user commands for making changes to the current program, with direct and immediate visual display of all changes.

If you are creating a new program, you will normally start PILOT and type "EDIT" to enter the edit mode. In this mode you can create PILOT program text within the computer's memory. When you leave edit mode and return to immediate mode, you may wish to SAVE the program you have created or type "RUN" to execute it.

Edit mode is described more completely in Section 5. Edit mode is not available to the user of PILOT that has been configured for an abbreviated student version.

SECTION 3

PROGRAM DEVELOPMENT

3.1 INTRODUCTION

A student is sitting at the computer keyboard. He is about to begin executing a PILOT program that he has been told consists of a reading comprehension exercise. You have shown him how to load the program, or he has read Section 2 of this manual and found the instructions for himself. He types LOAD <program name> <cr>, and soon sees on the screen the question:

> "Do you like to read? (Type in an answer.)"

He types , "Not much," and the computer replies:

> "You don't? Well, this exercise isn't too long."

Or he types , "Yes, most of the time," and the computer replies:

> "Oh, good. I hope you enjoy this exercise!"

The reading exercise program has done four things so far: 1) typed text, 2) accepted an answer, 3) matched that answer with some kind of vocabulary data, and 4) replied accordingly (by again typing text). Let us use a shorthand for the three basic functions involved:

T: for "type text"
A: for "accept an answer"
M: for "match"

Let us also suppose that, by appending a Y or an N to one of the codes, we can "condition" the performance of the function, i.e., we can make performance conditional upon a successful match of the student's last response with the vocabulary data. Here is the hypothetical program:

CODE	FUNCTION
T:Do you like to read?...	Type this text on the video display.
A:	Accept an answer from the keyboard.
M:no	Check for a match with this text.
TY:You don't? Well,...	Type this text if a match was found.
TN:Oh,good. I hope...	Type this if a match was not found.

The sequence of statements in the column labeled "CODE" is, as you have probably guessed, valid PILOT language.

A PILOT program consists of a series of statements that give a step-by-step description of what the computer must do to interact with a person sitting at a keyboard. An instruction code, consisting of one or more letters followed by a colon, defines the statement type and determines its function.

Single letter codes define "core" instructions that occur in all versions of PILOT. Of these, the three most important are the ones we have already used (T, A and M).

The letters Y and N are "conditioners" that can be appended to another code, causing it to be effective or not depending upon the success of the last attempted match.

PILOT is designed for the development of conversational programs that allow relatively free response by the user. As indicated in the example above, recognition of user responses is accomplished by searching the responses for specific words or word stems. In ordinary conversation, one or more words in a sentence often carry most of the sentence's meaning. The M statement is used to define those words, portions of words, or word groups that we want recognized in an answer. In the example we have been using, the M-statement will find the desired match in the answers "no," "not much," "nope," "not really," "I don't know," and "Not unless it's about sports." It will not respond properly to "only comic books," "very seldom," or "Sometimes."

Conversation in a PILOT program is facilitated both by the structure of the language, and by the programmer's ingenuity in designing questions and matching replies. For example, it is more effective to search for a negative answer than to search for a positive one, because the letter combination "no" will occur in almost all negative answers. It is permissible to use more than one word or word stem in an M: statement.

Detailed explanations of all of the statements in the PILOT language are found in Section 4.

3.2 CONDITIONAL EXECUTION, CONTINUATION LINES, LABELS

If we were limited to asking a question, scanning for a particular word in the answer, and replying accordingly, PILOT would not be a very useful language. Fortunately there are other statements and options that we can use to accomplish more complicated tasks. For our reading comprehension program, we want to be able to present a reading passage more than one line long, ask related questions, and, perhaps, move to one of several possible points in the program, depending upon the fulfillment of a certain condition. To do these things, we need provisions (some of which we have already discovered) for conditional execution, ways to continue text beyond the first line entered, and a way of labeling parts of a program so that control can be directed to any one of them by name. (To direct control is to determine which statement of the program will be executed next.) PILOT has all of these capabilities.

CONDITIONAL EXECUTION

As we have already seen, Y or N can be appended to any statement, as a "conditioner" of that statement. Operation will then depend upon whether the last M statement was successful. Y: by itself is a shorthand form of TY: Likewise, N: by itself is a shorthand form of TN:

The execution of a PILOT statement can also be made dependent upon the value of a numeric variable or expression; the variable or expression should appear in parentheses between the command code and the colon. (Rules for variables are given in Section 3.3; rules for expressions are given in Section 4, in the discussion of the Compute statement.) If the value of the variable or expression is greater than 0, then the statement will be executed. Otherwise it will be ignored. TY(X): will be executed if X=10, but not if X=0.

If both a numeric condition and a Y-N condition are used in the same statement, the statement will be executed only if both of the prescribed conditions are satisfied. For example, the statement TY(X): will type subsequent text only if the most recent match was successful AND the value of X is positive. (For instance, you can determine that a student may proceed to the next exercise only if 1) he has no questions on the previous exercise, and 2) his score on that exercise was acceptable.) T(X)Y: and TY(X): are interchangeable.

CONTINUATION

If you want to type a number of lines of text, only the first line must be preceded by a T: All subsequent lines may be preceded simply by a colon.

```
T:THIS TEXT WILL BE DISPLAYED
 :AND THIS WILL ALSO BE DISPLAYED
```

PILOT, Version 5 will halt and display unrecognized PILOT statements, but by putting colons in column one you can "draw" a picture by arranging characters on the screen:

EXAMPLE:

```
------------------------------------------------------------------
 (PILOT program)                     (display)

T: What shape is this?           What shape is this?
:           .                            .
:         .   .                        .   .
:       .       .                    .       .
:     .           .                .           .
:   . . . . . . . . .            . . . . . . . . .
A:                               (user enters "triangular")
M:triang
------------------------------------------------------------------
```

PILOT LABELS

Any PILOT statement may include a label, either as the first element of the statement or on the line above it. A label serves as a marker in the code, so that you can say in your program, "Go to this point right away, and do whatever you are told when you get there." (As we shall see, you can also say, "and be sure to come back, when you're finished!") PILOT labels begin with an asterisk (*) and end with a blank. They may have a maximum of ten characters. If a label is the only item in a line, it refers to the statement or group of statements following it.

No complaint is made if there are duplicate labels, but only the first in sequence will be found when the label is referenced.

EXAMPLE:

```
-----------------------------------------------------------------
 (PILOT program)                   (display)

    *LABEL
       T:DISPLAY THIS              DISPLAY THIS
           .....                   .....
           .....                   .....
-----------------------------------------------------------------
```

3.3 JUMPS AND SUBROUTINES

Now that we have some additional tools, we can expand and develop our reading exercise program. For the sake of convenience, let us say that the reading passage that we want to use is a popular fairy tale.

```
        T: Once upon a time....
           ....
           ....
         : And they lived happily ever after.
```

The student will be considering the text in screenfuls, that is, in segments with a maximum number of lines limited by the size of the screen. By inserting an A statement or FOOT statement after every screenful of information, we give the student an opportunity to indicate, by his response, that he has finished reading that portion of the text and is prepared to continue with the exercise. If we did not arrange for a pause, the text would appear and then disappear off the top of the screen much too fast for anyone to read it.

Thus, we might display the first part of "Beauty and the Beast," in which a father takes leave of his three daughters before setting out on a business trip. He asks each daughter, in turn, what she would like him to bring her ....

We have already used one easy method of programming question and answer sequences. Using the T, A, and M statements, we can ask a question and analyze the answer in either of the following ways:

```
T: What did the youngest daughter want her father
 : to bring her when he returned from his journey?
A:
M: rose
TN: No, she wanted him to bring her a rose.
```

or

```
T: What did the youngest daughter want her father
 : to bring her when he returned from his journey?
 : Enter the number of the correct response.
 :
 : 1) a necklace
 : 2) a rose
 : 3) a frog
A:
M: 2
TN: No, she wanted him to bring her a rose.
```

After the student has answered the first set of questions, we can give him another choice. We can use a T statement to ask him, "Do you want to read the last story again?" Presumably, if his answer is negative, we will want to go on to the next story; if his answer is affirmative, we will want to jump backward in the program and repeat execution of those statements that cause the story to be displayed.

To effect the jump, we need two elements of PILOT language: a label to which to direct the action of the program, and a statement that directs the action to that label. The PILOT statement that initiates a jump to a specified label is the J: statement. Let us go back and insert a label at the beginning of the original series of T statements:

```
*STORY1
T: Once upon a time...
 ....
 ....
```

and then let us insert the J statement, after the first group of questions:

```
 ....
 ....
T: Do you want to read the last story again?
A:
M: yes
JY: *STORY1
```

Notice the use of the conditioner as part of the J statement. With the program arranged this way, the student who asks to have the story repeated must answer all of the questions again. The J statement does not provide for any return to the original statement. What if you wanted the student to be able to review the story without answering the questions again or to look back at the story between questions to correct a wrong answer before proceeding? We need a way to isolate a group of statements to which we can jump from wherever we are, without losing our place in the execution of the program. (We also need a way to jump ahead in a program, without necessarily having to skip all of the intervening code.) A group of statements that we reference as a unit and after whose execution we automatically return to our prior position in the program, is called a SUBROUTINE. A PILOT statement that calls for the USE of a certain labeled group of statements as a subroutine is the U statement.

To use a U statement to enable our student to review what he has just read, we can mark the beginning and end of the set of statements that we want regarded as a subroutine. The label *STORY1, which we inserted to mark the destination of a J statement, will mark the beginning of the subroutine STORY1. The statement to mark the end of a subroutine, or of an entire program, is called the End, or E statement.

In our example, the E statement should be placed at the conclusion of the text:

```
*STORY1
 ....
 ....
T: And they lived happily ever after.
E:
 ....
 ....
```

The U statement can replace the current J statement, or it can be put after any one of the individual questions, or it can be inserted repeatedly, wherever you want the story to be displayed again. At any of these points, you can have the student "detour" to a vocabulary or grammar exercise within the same program by marking a subroutine accordingly and calling it. The subroutine can be called either unconditionally or depending upon fulfillment of a specified condition. The format of a U statement is identical to that of a J statement.

A subroutine may be used within another subroutine with the limitation that no more than seven U statements may be pending at once.

```
*STORY1
 ....
 ....
 T: And they lived happily ever after.
 E:
 ....
 ....
 T: Do you want to read the last story again?
 A:
 M: yes
 UY: *STORY1
 ....
 ....
```

3.4 PILOT VARIABLES

A variable was defined earlier as "a name to which different values can be assigned." In PILOT, we can concern ourselves with two types of variables: numeric variables, and string variables.

STRING VARIABLES

PILOT allows the input to an Accept (A:) statement to be saved as a character string and retrieved later in the program. The string variable name, with which the string will henceforth be associated, is written after the colon in the A statement. A string variable name begins with "$" and ends with a blank or carriage return.

Later on, if the string variable name appears in any portion of a T:, RW:, M:, J:, or E: statement, or to the right of = in a C: statement, it will be replaced with the value most recently assigned to it. If there has not been a value assigned to it, or if the variable name is preceded by "\", the variable name, itself, will be used by default.

EXAMPLE:

```
-------------------------------------------------------------------
   (PILOT program)                   (display)

T: What is your favorite story?  What is your favorite story?
A:$TITLE                         (User enters "Pinocchio")
T: Who is the hero in $TITLE?    Who is the hero in Pinocchio?
   ....
   ....
T: This is $unknown.             This is $unknown.
-------------------------------------------------------------------
```

A string variable name may have a maximum of ten characters in addition to the dollar sign. The maximum length of a character string to be stored as a string variable is dictated by the parameter INMAX, which can be set within a PILOT program and will be described in Section 4.

You can change the value of a string variable by accepting new contents in another A statement (thereby causing the new entry to supersede the former contents), or by assigning a new contents as part of a C: statement.

String variables are erased with the VNEW:$ command, and also by VNEW and NEW. VNEW:$ erases only string variables, VNEW erases both string variables and zeroes numeric variables, and NEW also erases the current program from memory and causes a complete reset to startup conditions.

The special meaning of "$" to denote a string variable name is cancelled if it is preceded by a backslash (\). Thus \$name will be replaced by $name and not the contents of $name.

NUMERIC VARIABLES

Numeric variables may be assigned integer, or whole number, values in the range of -32768 to 32767. Numeric variable names begin with the pound sign character (#) and consist of one letter (A-Z). Like string variable names, they are inserted after the colon in an Accept (A:) statement, so that whatever legal value is entered becomes associated with the named numeric variable. In the execution of an ensuing Type (T:) or Remark Write (RW:) statement, the current value of a variable will be substituted for the variable name. If no value has been assigned to the variable, the current value is assumed to be zero.

EXAMPLE:

```
-----------------------------------------------------------------
     (PILOT  program)                    ( display )

          A:#X                           "23" entered
          T:THE VALUE IS #X              THE VALUE IS 23
-----------------------------------------------------------------
```

You can change the value of a numeric variable by accepting a new value in another A statement (thereby causing the new entry to supersede the former value), or by assigning a new value as part of a Compute (C:) statement.

As described above for string variables, all numeric variables are reset to zero by the commands VNEW:#, VNEW, and NEW.

The special meaning of "#" to denote a numeric variable name is cancelled if it is preceded by a backslash (\). Thus \#n will be replaced by #n and not by the value of n.

3.5 COMPUTE AND REMARK

In our fairy tale example, we might want to dictate that a jump be made, a subroutine used, or a message displayed, depending on how well the student performed on the last set of exercises. In anticipation of this need and other arithmetic applications, PILOT provides a Compute (C) statement.

EXAMPLE:

```
     (PILOT program)                      (display)

T: How old are you?                 How old are you?
A:#N                                (User enters "6")
C: N=N+1
T: Then you'll be #N next year.     Then you'll be 7 next year.
```

To keep track of a student's score on an exercise, we can insert, after every M statement following a question, a command that adds 1 to the value of a numeric variable ON THE CONDITION that the previous answer was correct:

```
          ....
          ....
          T: What did the youngest daughter...
          A:
          M: rose
          CY: N=N+1
          ....
          ....
```

The C: statement has a generalized assignment capability for both string variables and numeric variables. Conditions governing the use of compute statements are discussed more explicitly in Section 4.

The final "core" statement in this initial overview of PILOT is the Remark (R:) statment, which might, at first glance, seem almost useless, since its function is to tell PILOT, "Ignore this remark." Actually, the R statement can be very useful: as you develop bigger and more complicated PILOT programs, you might want to write yourself or another programmer notes about what is happening in the code at a particular point. Say, for example, you have a very long program, so long that you have divided it into several PROGRAM SEGMENTS to be executed in succession. At the beginning of a certain segment, you might want to insert some documentation for yourself.

EXAMPLE:

```
         R: This is the second segment of an exercise program
         R: intended for students who performed below average
         R: on the diagnostic test.
          ....
          ....
```

3.6 IMMEDIATE OPERATION

Whenever PILOT is awaiting input from the keyboard, whether in response to an A statement, or at the PILOT restart point, the user is able to submit a command of his own for immediate execution. The format of such an "immediate command" is important: any statement within PILOT can be entered as a command, as long as it is preceded by a backslash (\) and, where there is not already a colon in the format of the statement, terminated by a colon (:). For example,"\U:*LABEL" will cause immediate execution of the referenced subroutine; \LOAD: will cause a new program to be loaded from the disk.

The interactive possibilities of PILOT can be greatly enhanced by the use of these immediate commands. For example, there is the option of letting the student specify a detour to the subroutine of his choice, or of allowing him to calculate his own score on an exercise, in order to determine whether to continue.

There are, however, times when the final user of a PILOT program should not be allowed to make changes to the program, or to change scores that might be collected. PILOT allows creation of a special student version that does not allow access to EDIT mode or to certain immediate commands.

After the execution of an immediate command, PILOT returns to the A statement in reply to which the immediate command was entered and awaits an answer. (The J and U commands entered for immediate execution work slightly differently, in that the answer to the last question must be supplied for the immediate command to take effect. In the other cases, the user will be expected to answer the last question only AFTER the execution of the immediate command.) Exceptions to the rule of return to the A statement are the \LOAD: and \GET: commands, whose execution obliterates the program in memory, and \<cr>, which stops the current program and returns PILOT to immediate mode.

When PILOT begins operation, it displays the message "READY". At this point of control, which we will call "PILOT restart," a number of commands may be entered for immediate operation without an initial backslash (\) or a terminal colon (:). The immediate command "?" will display their names as a reminder. They are: LOAD, GET, RUN, DIR, BYE, INFO (available in both student and author mode), and SAVE, CREATEF, KILLF, LIST, NEW, VNEW, SET, EDIT (available in author mode only).

For example, at PILOT restart, typing RUN will cause program execution to begin. Thus, if you have just entered a program in the PILOT EDITor (see section 5), you will initiate the execution of that program. Otherwise, you will execute the last program stored (using LOAD or GET) in the program buffer (unless you left PILOT in the meantime).

SECTION 4

PILOT STATEMENT DESCRIPTIONS

4.1 STATEMENT SUMMARY

The following section describes each of the PILOT instructions provided in this version of PILOT for the 8080 microprocessor. The descriptions include format, function, and examples. Where appropriate, error messages that might be generated by invalid statements or invalid user response are described. In the examples below, the column marked "display" shows text and messages IN THE ORDER THAT THEY APPEAR ON THE SCREEN. This order does not necessarily correspond to the order of statements in the PILOT program.

We have divided PILOT instructions into six categories, corresponding, roughly, to the type of instruction or to the service performed. Tne groupings are:

Sect. 4.2) Core instructions--those which perform all of the most basic functions in PILOT (already discussed).

4.3) Cursor and video control instructions--those which enable you to determine where text will appear on the video screen,

4.4) Aids to conversation--those instructions which facilitate dialogue in PILOT by extending the available core instructions,

4.5) Instructions that set various kinds of parameters

4.6) File manipulation instructions--those related to storing and retrieving programs and data, and

4.7) Commands usually used in immediate mode from PILOT restart. (They may actually be executed from within a PILOT program, but might cause complications if not used carefully.)

Core instructions are limited to single letter codes. These instructions are standard in format and operation for many different implementations of PILOT. It is therefore advantageous to use core instructions as much as possible, so that users of other versions of PILOT may benefit from your work. Multi-letter codes represent "keyword" instructions that have been added to PILOT to meet special needs. When you use them, you should keep in mind that they do not necessarily exist in all other versions of PILOT.

4.2 CORE INSTRUCTIONS

These are the statements that you will be using most frequently in your PILOT programming. They let you type text on the video screen, accept and analyze responses, and alter the order in which statements are executed. There are a number of "keyword" instructions, which are discussed in connection with with the core instructions to which they are closely related. These are TH:, RW:, and the variations on J: and M:. Remember that such instructions do not share the standardization of normal core instructions.

STATEMENT: TYPE (T:), YES (Y:), and NO (N:)
TYPE AND HANG (TH:)
TYPE TO PRINTER (TP:)

FORMAT:

```
--------------------------------------------------------------------
        [<label>] T [<cond>] : <message>
        [<label>] Y [<cond>] : <message>
        [<label>] N [<cond>] : <message>
        [<label>] TH [<cond>] : <message>
        [<label>] TP [<cond>] : <message>
--------------------------------------------------------------------
```

DESCRIPTION: Display a message to the PILOT user. A message consists of a character string that may include one or more variable names. All character positions to the right of the colon are reproduced literally except that the values of variables are inserted as replacements for their names. String variable names are a maximum of ten characters long and are preceded by "$." Numeric variable names are one letter long and are preceded by "#."

```
        $abacus is a string variable.
        #a is a numeric variable.
```

The backward slash "\" immediately preceding "$" or "#" disables their special function, causing them to be displayed. The "\" itself does not print in this context.

In most cases it is not necessary to provide an explicit terminator to a string variable name in order to produce appropriate replacement of labeled text. There are, however, some instances that require a terminator to avoid confusion, and the tilde ("~") may be used for this purpose. It will neither print nor cause a space in the output. Another use is to terminate blanks in a T statement used to erase existing text.

Multiple "$" prefixes in string variable names cause an indirect reference to the string variable. For example, "$$A" indicates that "$A" contains the string variable NAME (the name of "A") rather than the string variable itself.

YES (Y:) and NO (N:) statements are abbreviated forms of TY: and TN: and are entirely equivalent in operation. It is acceptable to use the arithmetic conditional with either of these forms. For example, Y(X): will be executed on the condition that 1) the last match was successful, AND 2) the value of X is positive.

The TH statement is like the T statement, except that the typing position "hangs" after the message is displayed. The usual progress to the next line is suppressed, so that the user's next response is displayed immediately after the message on the screen, rather than on the next line. Thus,

```
        TH: 6 + 7 = _
        A:
```

allows the student to type his answer where it seems most natural for the answer to appear. An underline may be used to position the cursor to the right of the message.

The TP statement is the same as the T statement, except that the message element is directed to the list device instead of to the console. As one application, it has been successfully used for operation of the Votrax "Type-N-Talk" device.

ERROR MESSAGES: Reference to a string variable that has not yet been assigned a value will cause display of the string variable name,including the "$." You may deliberately arrange for a string variable name, rather than a value, to appear. If the problem arises unexpectedly, it is usually caused by a misspelling of the variable name. A numeric variable to which no value has been assigned is displayed with a value of zero.

EXAMPLES:

```
---------------------------------------------------------------
     (PILOT program)                  (display)

*START
T: Please tell me your name.     Please tell me your name.
A:$NAME                          (user enters "Alex")
T: Hi, $NAME!                    Hi, Alex!
 : How old are you?              How old are you?
A:#a                             (user enters "8")
T:Is it fun to be #a ?           Is it fun to be 8 ?
---------------------------------------------------------------

C:$NAME=MIKE
C:$MIKE=MICHAEL
T:$NAME                          MIKE
T:$$NAME                         MICHAEL
T:$$$NAME                        $MICHAEL
---------------------------------------------------------------
```

STATEMENT: ACCEPT AN ANSWER (A:)
ACCEPT AND HANG (AH:)
ACCEPT SINGLE CHARACTER (AS:)

FORMAT:

```
------------------------------------------------------------------
        [<label>] A [<cond>] :
        [<label>] A [<cond>] : $<string variable>
        [<label>] A [<cond>] : #<numeric variable>
        [<label>] A [<cond>] : =<string variable>
        [<label>] AH [<cond>]:
        [<label>] AS [<cond>]:
------------------------------------------------------------------
```

DESCRIPTION: Makes it possible for the user to communicate with PILOT from the keyboard. While entering a response, the user may cancel the last character entered by depressing the DELete key or the backspace key, and cancel the current line by depressing ctrl-X. ("!" is displayed on the screen whenever ctrl-X is used.) A carriage return signals the termination of a line. If the line being entered exceeds the maximum length allowed for a response (i.e., if the present or default value of the parameter INMAX is exceeded), PILOT supplies its own carriage return and regards the line as terminated.

If there is nothing to the right of the colon in an A statement, the response will be retained in a special "accept buffer," so that it may be considered by subsequent Match and WRite commands (see later).

If a variable name is used, then the response will be stored as a value for that variable.

If an "=" operator is present, then the contents of the string variable to the right of the "=" will be assigned to the accept buffer, rather than the normal input from the keyboard.

If a numeric variable name is included and a non-numeric response is entered, an error message will be displayed and another response will be accepted.

The AH statement is like the A statement, except that the typing position "hangs" after the response is displayed. For example, a following T statement will appear on the same line after the user's response, rather on the next line.

The AS statement accepts a single character from the keyboard and proceeds without waiting for "RETURN" to be pressed.

LIMITATIONS: The length of a response is limited to 80 characters or to a lower limit set by the keyword statement INMAX. (The ormat of the INMAX statement is given in Section 4.5.) PILOT. A numeric variable response must be an integer between -32768 and 32767.

ERROR MESSAGES: "*NUMERIC RESPONSE REQUIRED" occurs if a non-numeric entry is attempted in response to a command requesting a value for a numeric variable. The user is expected to enter another response.

"*NO ROOM" indicates that the area available in memory for string variable storage has been exhausted. (See discussion of VNEW$ for some ideas about how to cope with this problem.)

EXAMPLE:

```
    (PILOT program)                  (display)

     T:WHO ARE YOU?                 WHO ARE YOU?
     A:                             (user enters "your son")
     .....                          .....
     .....                          .....
     T:WHAT IS YOUR NAME?           WHAT IS YOUR NAME?
     A:$NAME                        (user enters "Timothy")
     .....                          .....
     .....                          .....
     T:WHAT IS YOUR AGE?            WHAT IS YOUR AGE?
     A:#A                           (user enters "10")
     .....                          .....
     .....                          .....
```

STATEMENT: MATCH (M:)
MATCH JUMP (MJ:)

FORMAT:

```
    [<label>] M  [<cond>] : <pattern>[,<pattern>...,<pattern>]
    [<label>] MJ [<cond>] : <pattern>[,<pattern>...,<pattern>]
```

DESCRIPTION: The response received by the last A statement is scanned for occurrences of ANY of the character patterns that follow the colon in the Match statement. If the response is found to contain one or more such strings, a subsequent statement including a Y condition will be executed. In this form of the M statement, commas are used to separate patterns in the list, with blanks considered part of the pattern (except those following the last item. (Any response is regarded as as though it begins with a blank, however, so that the statement M: YES will match a "yes" response to the previous question, whether or not the user preceded the response with a blank.) If you want to specify a pattern ending with a blank as the last pattern in your list, follow that pattern with a comma, as in the third example below. Contiguous blanks are reduced to one. For example, the pattern "M r." will match with "M r."   "Mr." will match with neither "M r." nor "M r.", however. Here is a more technical description of how the Match statement operates:

1. Each pattern in the M statement has multiple blanks reduced to one.
2. The user's last response has a blank added to each end.
3. The user's last response has multiple blanks reduced to one.
4. A moving window scan of the response is made with each pattern, until either a match is found or the input is exhausted. Upper/lower case is ignored.

If the first character after the colon is a vertical bar ("|"), then that character will be the field separator for the match list, instead of a comma. If the first character after the colon is anything other than a vertical bar, then the field separator will be a comma.

The question mark ("?") in an M-item will match any single character (or no character) in that position, and the asterisk ("*") will match any number of characters in that position. Placing "\" immediately preceding "?" or "*" allows a search for those characters.

Three string variables are produced as a result of a successful match. $LEFT contains as its contents everything to the left of the matched data, $MATCH contains the matched data, and $RIGHT contains everything to the right of the matched data. They may be used like other string variables. If the match was unsuccessful, the three variables are left with their prior contents.

The MATCH JUMP statement (MJ:) causes a jump to the next M, MC, or MJ statement if the present attempt to match a response is unsuccessful. Look at the example in the discussion of the Jump statement MJ: YES would be exactly equivalent in execution to M: YES followed by JN:@M (see under JUMP, p. 4-9.)

EXAMPLES:

```
                  M:A,B,C
                         Matches A or B or HAT or ALICE or JOB
                         Does not match TENT or X

                  M:  A, B, C
                         Matches A or B or ALICE
                         Does not match JOB or HAT

                  M:  A , B , C ,
                         Matches only A or B or C
```

```
                  T:PLEASE TYPE A TEST SENTENCE

          User types:     "HERE IS A TEST SENTENCE."

                  M: WAS , IS , WERE ,

          The contents of $LEFT   = "HERE"
                 "        $MATCH  = " IS "
                 "        $RIGHT  = "A TEST SENTENCE."
```

STATEMENT: JUMP (J:)
JUMP ACCORDING TO MATCH (JM:)

FORMAT:

```
----------------------------------------------------------------
     [<label>] J   [<cond>] : [*] <destination label>
     [<label>] J   [<cond>] : @M
     [<label>] J   [<cond>] : @P
     [<label>] J   [<cond>] : @A
     [<label>] JM: [<cond>] : [*]<label1>,[*]<label2>,...
----------------------------------------------------------------
```

DESCRIPTION: Causes a jump to the specified destination in the current program. The most common form of the statement is that which results in a jump to a particular label. (The asterisk is optional.)

The next two forms of the statement, illustrated above, cause a jump to the next Match statement (M, MJ, MC), or PR statement, respectively. The PR statement is described in section 4.4. In general, it serves only as a destination for a J or U statement.

In case it is not immediately evident why one would want to use the alternative forms, let us consider an example. The student has just finished reading a passage about color and color groupings. As a test of his comprehension, we ask him to name a warm primary color. We want to give him an opportunity to review any material that he does not seem to understand. Thus we want not only to check whether his answer is correct, but to determine in what respect his answer is incorrect, and proceed accordingly. He need not be asked to review what he already understands. Using the different forms of the Jump instruction, we can generate the code in the example below.

The J:@A statement will jump back to the last A statement without the necessity of that statement having a label. It will most often be used in a sequence such as

```
A:
M:item1,item2
TN:No, that's not right.  Try again.
JN:@A
```

The Jump according to Match statement (MJ:) jumps to one of a list of labels according to which in the last preceding list of M-items was matched. If the third M-item was matched, then the jump is to the third label. If no item is matched, then there is no jump.

The Match Jump statement (MJ:) provides a shorthand for the sequence of Match (M:) followed by JN:@M. The format is shown under our description of the Match statement (M:).

If the use of the J statement creates an endless loop, remember that "\" (Return) may be typed at any A statement to return to immediate mode. If the loop does not contain an A statement, pressing the ESC key will provide an exit to immediate mode.

ERROR MESSAGES: The designation for a destination that can not be found will be displayed, followed by:

"-NOT FOUND"

EXAMPLE:

```
-----------------------------------------------------------------
   (PILOT program)                     (display)

    T: Name a warm primary color.    Name a warm primary color.
*TEST1 A:$color                      (user enters "green")
    M: red,yellow,blue               (match not found)
    JY:@M
T:$color is not a primary color;    Green is not a primary color;
:Let's review the primary colors.   Let's review the primary ....
     ....                            ....
     ....                            ....
     : Now name a primary color.     Now name a primary color.
    J:*TEST1                         (user enters "blue")
    M:blue
    JN:@P
T: $color is not a warm color;       Blue is not a warm color;
:Let's review the warm and cool      Let's review the warm.....
:colors.
     ....                            .....
     ....                            .....
    PR:
     ....                            .....
     ....                            .....
-----------------------------------------------------------------
```

STATEMENT: USE SUBROUTINE (U:)

FORMAT:

```
------------------------------------------------------------------
       [<label>] U [<cond>] : [*]<destination label>
       [<label>] U [<cond>] : @M
       [<label>] U [<cond>] : @P
------------------------------------------------------------------
```

DESCRIPTION: Causes a jump to a specified subroutine in the current program, and returns control, after the execution of that subroutine, to the statement following the Use instruction.

The end of a subroutine is marked by an End (E:) statement; the beginning of a subroutine is indicated by the label or instruction to the right of the colon in the Use statement.

Usually, the destination of this kind of jump is a label (which begins with an asterisk). The other possible destinations are @M and @P, just as in the J statement.

If you neglect to mark the end of a subroutine, the next End statement in your program will be regarded as the subroutine terminator. If the next End statement is the final statement in your program, control will return to the statement following the Use. Therefore, it is a good idea to be careful about marking subroutines, and to check for this kind of error when you find that some unlikely portion of your program is being repeated. (Consider what must happen if your U statement occurs between the desired subroutine and the next E statement in your program: the calling statement will be encountered AS PART OF THE SUBROUTINE, and will continually reinitiate the execution of that subroutine!)

LIMITATIONS: A subroutine may be used within another subroutine with the limitation that no more than seven Use statements may be pending at once. (The example shown for the E statement has a maximum of three Use statements pending at any particular point in the program.)

ERROR MESSAGES: The name of a nonexistent destination will be displayed, followed by:

"-NOT FOUND"

If too many subroutines are pending, you will receive the message:

"*USE DEPTH EXCEEDED"

If a subroutine is being re-executed within itself because the programmer failed to include a subroutine terminator (see above), the Use depth will eventually be exceeded.

```
EXAMPLE:
-----------------------------------------------------------------
     (PILOT program)                 (display)

*START  T:THIS IS WHERE WE START.   THIS IS WHERE WE START.
     T:DO YOU NEED INSTRUCTIONS?    DO YOU NEED INSTRUCTIONS?
     A:                             (user enters "yes")
     M:  YES                        (match found)
     UY:*INSTR                      (use subroutine INSTR)
       .....                         .....
       .....                         .....
*INSTR
T:THIS IS WHAT YOU NEED TO KNOW   THIS IS WHAT YOU NEED TO KNOW
       .....                         .....
       .....                         .....
     E:                             (return from subroutine)
-----------------------------------------------------------------
```

STATEMENT: END OF SUBROUTINE OR PROGRAM (E:)

FORMAT:

```
              [<label>] E [<cond>] :
```

DESCRIPTION: Indicates the end of a subroutine or the end of the current program. The first E statement encountered during the execution of a subroutine will be regarded as the end of that subroutine, and will return control to the statement following the calling (Use) statement for that subroutine. If an End statement is encountered with no Use statement pending, control is returned to PILOT restart.

Upon termination of the PILOT program, a data collection file left open will be closed.

LIMITATIONS: A maximum of seven subroutines may be be in execution at a given point in a PILOT program. See the error message in connection with the Use statement.

EXAMPLE:

```
      (PILOT program)         (display)
       *START  U:*FIRST
               T: TIME          NOW
       *END    E:               IS
       *FIRST  U:*SECOND        THE
               T: THE           TIME
               E:
       *SECOND U:*THIRD
               T: IS
               E:
       *THIRD  T: NOW
               E:
```

STATEMENT: COMPUTE (C:)

FORMAT:

```
-------------------------------------------------------------------
  [<label>] C [<cond>] : <num variable> = <num expression>
  [<label>] C [<cond>] : <str variable> = <text expression>
-------------------------------------------------------------------
```

DESCRIPTION: Assigns the value of a numeric expression to a numeric variable or assigns the value of a text expression to a string variable. The expression to the left of the equals sign must be either a numeric variable name or a string variable name.

If it is a numeric variable name, the initial "#" is optional. It may or may not be a variable to which a value has already been assigned in the PILOT program. (If there has been no such assignment in your program, the initial value is zero.) The expression to the right of the equals sign can consist of numbers, numeric variables (# optional) and operators (see below).

If it is a string variable name, the initial "$" is required. The expression on the right of the equals sign can consist of any combination of text, string variable names, and numeric variable names that will cause replacement of values in place of names as in the T statement.

ARITHMETIC OPERATORS

The arithmetic operators in PILOT are + (add), - (subtract), * (multiply), / (divide) and % (return the remainder of a division). There are some general rules to remember when you use these operators:

1) In PILOT there are no fractions or decimals. All numbers containing fractions or decimals are truncated to their integer part. 4 divided by 5 is zero, -10 divided by 3 is -3.

2) The % operation returns only the remainder of a division. If you wanted to execute a certain statement on condition that n be an odd number, you could write:

```
          ....
          ....
          C: A= n % 2
          T(A):#n is an odd number.
```

   because if n is odd, the remainder of the division of n by 2 will be greater than zero.

3) The order of operations is very important. In a numeric expression containing multiple operations:

a) The unary negative (that which reverses the sign of a number by negating it, as in the expressions -A and -15) is evaluated first.

b) Anything in parentheses is evaluated second. Within parentheses, the normal order of operations applies.

c) Multiplication, division, and the return-remainder operation are evaluated third.

d) Addition and subtraction are evaluated last.

In the absence of parentheses, operations of the same rank are performed from left to right. For example, 2*2/3 yields the result 1, not 0.

4) Neither the result of a computation nor any of its elements may be outside the acceptable range of integers -32768 to 32767; and attempt to store too high or too low a value will result in an error message and the assignment of a value of -32768 or 32767 (whichever is closer to the number indicated). For example, the statement

```
C: B = -n%3
```

will generate an error message if n is an expression that evaluates to -32768 (because 32768 is outside the acceptable range).

LOGICAL AND RELATIONAL OPERATORS

In addition to the arithmetic operators, PILOT allows the use of logical and relational operators in expressions. The result of a logical or relational operation is 1 if the condition is true, 0 if the condition is false. The logical and relational operators are:

\ (not)

This operation results in a value of 1 if the operand is equal to zero, 0 if the operand is not equal to zero. For example, the value of \6 is 0; the value of \0 is 1.

& (and)

This operation results in a value of 1 if both operands are non-zero, 0 if either or both operands equal zero. For example, the value of -3&20 is 1, because both operands are non-zero; the value of 0&14 is 0, because one of the operands is equal to 0.

! (or)

This operation results in a value of 1 if either operand is non-zero, 0 if both operands are equal to zero. For example, the value of 26!0 is 1; the value of A!0 is 0, if A is equal to zero.

= (equal to)

This operation results in a value of 1 if the operands are equal, 0 if they are not equal. For example, the value of 3+2=5 is 1; the value of 3+2=6 is 0.

<> (not equal to)

This operation results in a value of 1 if the operands are unequal, 0 if they are equal. For example, the value of 3+2<>5 is 0; the value of 3+2<>6 is 1.

< (less than)

This operation results in a value of 1 if the first operand is less than the second, 0 if the first operand is greater than or equal to the second. For example, the value of 14<7+8 is 1; the value of 382<5 is 0.

<= (less than or equal to)

This operation results in a value of 1 if the first operand is less than or equal to the second, 0 if the first operand is greater than the second. For example, the value of 3<=10 is 1; the value of -1<=-8 is 0.

> (greater than)

This operation results in a value of 1 if the first operand is greater than the second, 0 if the first operand is less than or equal to the second. For example, the value of 28>-28 is 1; the value of 6/4>6/3 is 0.

>= (greater than or equal to)

This operation results in a value of 1 if the first operand is greater than or equal to the second, 0 if the first operand is less than the second. For example, the value 5>=4+1 is 1; the value of 63>=32000 is 0.

Logical and relational operators are used primarily for conditional expressions. For example, if you write either

```
     U (A>6): *STEP1     or     C: B=A>6
                                U (B): *STEP1
```

STEP1 will be executed only if A is greater than 6.

If an expression contains both arithmetic operators and logical or relational operators, the full order of precedence is as follows (operators on the same line have equal rank):

unary + -	Operators of equal rank are evaluated left to right.
* / %	
+ -	
= <> < <= > >=	
\	
&	
!	

RANDOM NUMBER FUNCTION

The expression RND(n) represents a random number in the range 1 to n, with n having a maximum value of 32767.

LIMITATIONS: The expression to the left of the equals sign must be a numeric variable name, with the # omitted; it may or may not be a variable to which a value has already been assigned in the PILOT program. (If there has been no previous assignment within your program, the initial value is zero.) Neither the result of a computation, nor any of its elements, may be outside the acceptable range of integers -32678 to 32767: an attempt to store too high or too low a value will result in the assignment of -32768 or 32767, (whichever is closer to the number indicated).

ERROR MESSAGES: An offending expression is displayed, followed by the message:

"*CANNOT EVALUATE THE EXPRESSION"
or

"*VALUE OUT OF RANGE(-32768 TO 32767)"

An inability to evaluate the expression can occur as a result of bad format or in response to an illegal numeric variable name. After the message is displayed, PILOT awaits an entry from the keyboard. If <cr> is entered, execution proceeds with the next PILOT line. (You can alternatively supply an immediate command.)

The second message occurs if there is an attempt to set a value that is either too high or too low to be acceptable in PILOT (not between -32768 and 32767).

EXAMPLES:

```
--------------------------------------------------------------
      (PILOT program)             (display)

      C: x=(a+b)*(b-c)             (computations are made)
      C: q=(RND(n)+a)%10
--------------------------------------------------------------
```

STATEMENT: REMARK (R:)
REMARK WRITE (RW:)

FORMAT:

```
--------------------------------------------------------------
    [<label>] R [<cond>] : <text>
    [<label>] RW [<cond>] : <file number expression> <text>
--------------------------------------------------------------
```

DESCRIPTION: The Remark statement is a way of including useful descriptive information in the program listing without any effect on operation of the program. A Remark may be placed at any point in a PILOT program.

How easy was it to read the code given in relation to the Jump statement? Does the insertion of R statements make any difference?

EXAMPLE:

```
--------------------------------------------------------------
   (PILOT program)                         (display)

   R:Test on Color      (Same as for code in Jump section)
T:
:Name a warm primary color.
*TEST1
A:$color
   R:Is $color primary?
M:red,yellow,blue
   R:If student named a primary color, jump to the next test.
JY:@M
   R:If control reaches this point, $color is not primary.
   R: Review lesson.
T:$color is not a primary color;let's review primary colors.
:Now name a primary color.
   R:Repeat the first test.
J:*TEST1
   R:The color is primary. Test for warm or cool.
*TEST2
M: blue
   R: If there is no match, then the student has named a warm
   R: primary color. Proceed to the next problem.
JN:@P
   R: $color is not warm. Review lesson.
T:$color is not a warm color;let's review warm and cool colors.
PR:
--------------------------------------------------------------
```

There are only two differences between this program and the earlier version of it. The differences are 1)that the second version might be a bit easier to read, and 2)that the second is somewhat longer than the first. Both of these factors must be taken into consideration when you write a program. In the example above, we have probably incorporated more comments than are really necessary. A user who already knows PILOT will not need very many of these. It is also important not to clutter a program with unnecessary material, because the length of a program is limited by the capacity of the computer. Every character of Remark, although PILOT will not act on it in any way, contributes to the length of the program.

The RW statement is a Remark statement that writes remarks on a file. The RW statement is similar to the T statement, in that it involves a string of text, with substitution of current values for any string or numeric variables included in the text. The major difference is that whereas the T statement types text on the screen, the RW statement writes text into an opened data collection file on disk.

4.3 CURSOR AND VIDEO CONTROL INSTRUCTIONS

These instructions control the presentation of text on the video display. Recall the following two concepts introduced in Section 1.5:

1) The cursor marks the current position on the screen, and,

2) The screen may be divided into horizontal rows and vertical columns.

The position of the cursor on the video screen is called its "address."

The cursor control statements are:

STATEMENT:
CURSOR ADDRESS (CA:r,c)
CLEAR AND HOME (CH:)
CLEAR TO END OF LINE (CL:)
CLEAR TO END OF SCREEN (CE:)

FORMAT:

```
-------------------------------------------------------------------
              [<label>] CA [<cond>] : [<row>] [,<col>]
              [<label>] CH [<cond>] :
              [<label>] CL [<cond>] :
              [<label>] CE [<cond>] :
-------------------------------------------------------------------
```

DESCRIPTIONS: For the purpose of these commands, the first row on the screen is designated row 1, and the first column is designated column 1.

CA: sets the cursor address by row and column at which the next text is displayed or the next input is accepted. The row and column may be indicated by either an integer constant or a numeric variable name. If column indication is omitted, it is set to 1, and if row is omitted, it is set to the last used row + 1. To omit the row designation, follow the colon with a comma and your desired column number. A statement of the form CA:6 moves the cursor to row 6, column 1; a statement of the form CA:,6 moves the cursor down one line, and positions it at the sixth column of that line.

CH: clears the screen and sets cursor address to row 1 and column 1.

CL: clears from the current cursor position to the end of the current line. The cursor address is not changed.

CE: clears from the current cursor position to the end of the screen. The cursor address is not changed.

EXAMPLES:

(PILOT program)	(display)
CA:2,n	(causes the next text to begin in the nth column of the second row on the screen)
CA:,6	(causes the next text to begin in the sixth column of the next row on the screen.)

NOTE:

> None of these "cursor control" statements make the cursor appear as a character on the display screen. Outside of the the PILOT editor (section 5), it is probably best to consider the cursor as a position, ratner than as a visible marker of that position.

In addition to providing control of the cursor address and making it possible to clear all or portions of the screen, PILOT provides an easy way to emphasize portions of text with reverse video (reversal of black and white in the display).

The TH statement allows the user's next keyboard entry to appear immediately following the given line of text, instead of on the next line. There is an example in Section 4.1, where the T and TH statements are described.

STATEMENT: ROLL SCREEN (RL:n)

FORMAT:

```
[<label>] RL [<cond>] : <expression>
```

DESCRIPTION: Causes screen contents to roll, or scroll, upward a specified number of lines. The expression indicates the desired number of lines. It may be an integer constant, a numeric variable name, or an expression that will evaluate to an integer value. If no value is given, the screen is rolled one line.

With each roll, the contents of the top line are lost, and a blank line appears at the bottom of the screen.

The RL statement is occasionally useful to retain information presented in the lower portion of the screen while opening an area to present new information or a question.

EXAMPLE:

(PILOT program)	(display)
RL:8	(results in an 8-line upward roll)

4.4 AIDS TO CONVERSATION

There are a number of PILOT statements that make it easier to converse within the PILOT system. They are FOOT, PAUSE, VNEW, PR, CALL, XI, and XS.

STATEMENT: FOOT OF SCREEN HALT AND PROMPT (FOOT:)

FORMAT:

```
          [<label>] FOOT [<cond>] : [<text>]
```

DESCRIPTION: Causes the text provided after the colon to appear on the bottom line of the screen and executes an A statement. If there is no text after the colon, the message is:

'PRESS "RETURN" TO GO ON..'

Execution will resume when the user hits the RETURN key. (Even if the message is not "PRESS RETURN TO GO ON.")

When we thought about writing the reading exercise program, two of the problems were how much text could be displayed on the screen at once, and how to give the student enough time to read it. We decided to display a screenful of text and then ask a question at the bottom of the screen. FOOT is a convenient combination of three commands:

```
CA:16      (number of the last line on screen)
T: <text>
A:
```

A possible reason to use FOOT, as opposed to the three other statements, is that FOOT takes up less space in the program buffer; a disadvantage is that FOOT is not included in all versions of PILOT.

EXAMPLE:

```
      (PILOT program)          (display)
       ....                     ....
       ....                     ....
T:Now we'll go on ...       Now we'll go on...
  FOOT:                     PRESS "RETURN" TO GO ON
                            (at last line position)
T:NEXT QUESTION:            (user enters <cr>)
       ........             NEXT QUESTION
```

STATEMENT: PAUSE (PAUSE: or PA:)

FORMAT:

```
[<label>] PAUSE [<cond>] : [<expression>]
[<label>] PA    [<cond>] : [<expression>]
```

DESCRIPTION: Causes program operation to halt for a specified length of time, 1 to 99 seconds. The optional expression indicates the desired time. It may be an integer constant, a numeric variable name, or an expression that will evaluate to an integer value. If no time is given, the wait is approximately one second. (The figures given are for a 2 mh. 8080 Computer. Computers with higher speed processors will need an adjustment to the pause timing factor PATIM listed in the NVPILOT.PRN file).

The PAUSE statement permits you to regulate the amount of time a student has to study text on the video screen. This feature of PILOT has an obvious application to timed exercises of various kinds. Consider how you could change the reading exercise program, if you did not want to give the student unlimited time to study each screenful of material.

EXAMPLE:

(PILOT program)	(display)
PAUSE:3	(results in a 3 second pause)

Section 4.4: Aids to conversation.

STATEMENT: NEW VARIABLES (VNEW:$, VNEW:#, or VNEW:)

FORMAT:

```
------------------------------------------------------------------
                [<label>] VNEW [<cond>] :$
                [<label>] VNEW [<cond>] :#
                [<label>] VNEW [<cond>] :
                VNEW                          (at PILOT restart)
------------------------------------------------------------------
```

DESCRIPTION: The VNEW:$ statement clears all string variables defined earlier in the program, and thereby reclaims the storage space that these occupied. You can use the VNEW:$ statement to avoid cluttering the program buffer with unnecessary data whenever you have accumulated a lot of variables no longer in use.

One of the most important conversational features of PILOT is the ability to retain a user's response, so that it can be retrieved later in the program. The VNEW:$ statement lets the programmer deal with potential problems caused by the limited storage capacity of the computer.

The VNEW:# statement resets the value of all numeric variables to zero. Note that the space-consuming problem of string variables does not apply to numeric variables. Each value assigned to a numeric variable actually replaces the old one, so that there are always 26 numeric variables, corresponding to the letters of the alphabet, stored in the memory of the computer. Each of these numeric variables always has a value (assumed to be zero where no other has been assigned) and none of them are stored with the program in the program buffer area. Because of these distinctions, there is no storage problem with numeric variables. Such values are initialized, i.e., set to zero, rather than actually cleared from memory, in response to a VNEW:# , VNEW:, or NEW command.

As you may have guessed, VNEW: (without $ or # after the colon), clears both string variables and numeric variables. NEW is used in immediate mode to return PILOT to its initial startup condition. NEW is discussed in Section 4.7, below.

EXAMPLE:

```
------------------------------------------------------------------
    (PILOT program)                     (display)

  T:GIVE YOUR ANSWER NOW.         GIVE YOUR ANSWER NOW.
 *ANSWER A:$TEXT                  (user enters "jhgblj")
    M:PATTERN                     (match not found)
    N:SOMETHING'S WRONG.          SOMETHING'S WRONG.
    N:PLEASE TRY THAT AGAIN.      PLEASE TRY THAT AGAIN.
    VNEWN:$                       ....
    JN:*ANSWER                    ....
   T:FINE, LET'S GO ON.           FINE, LET'S GO ON.
     .....                        .....
------------------------------------------------------------------
```

STATEMENT: PROBLEM START (PR:)

FORMAT:

```
-----------------------------------------------------------------
                [<label>] PR:
-----------------------------------------------------------------
```

DESCRIPTION: Provides a destination for the J:@P or U:@P command.

The PR: statement is very much like a label, in that it initiates no activity but functions as a landmark to be considered by other operations. It can also have the mnemonic advantages of a label, insofar as it can designate the beginning of the next exercise, set of questions, or general phase of a program. There are two reasons that you might decide to use a PR statement, rather than a label:

1) Unless you use labels only two characters long, PR: statements are shorter.

2) By using PR: to signify the beginning of a new section or procedure, you can avoid cluttering your program with a lot of extra labels that don't have very much significance. (You might want to label every SET of questions, but you probably would not want to label every question.) Using too many labels can be as confusing as using none.

STATEMENT: CALL PROGRAM ELSEWHERE IN MEMORY (CALL:)

FORMAT:

```
[<label>] CALL: <address> [,<argument>]
```

DESCRIPTION: Calls the assembly language routine whose address in memory is indicated by the first parameter following the colon, and stores an argument, if one is given, in registers D and E. The address must be a decimal number; the argument must be a number greater than -32768 and less than 32767. This statement permits you to call an assembly language program that you have stored in memory outside the PILOT program buffer. If the assembly language program includes a RETurn instruction, control will return to the PILOT program statement which immediately follows the CALL.

EXAMPLE:

(PILOT program)	(function)
....	....
....	....
CALL: 16385,-25	(executes a program at address 16385 and puts a value of -25 into registers D and E)

Section 4.4: Aids to conversation.

STATEMENT: EXECUTE IMMEDIATE (XI:)

FORMAT:

```
------------------------------------------------------------------
                 [<label>] XI [<cond>] : <string variable>
------------------------------------------------------------------
```

DESCRIPTION: Causes immediate execution of the contents of the string variable as a PILOT statement.

You can make use of an A statement or a C statement to assign contents to the string variable.

EXAMPLE:

```
------------------------------------------------------------------
      T:Type the name of the lesson you want.
      A:$LESSON
      C:$FILE = LOAD:$LESSON
      XI:$FILE
------------------------------------------------------------------
```

Section 4.4: Aids to conversation.

STATEMENT: EXECUTE SYSTEM (XS:)

FORMAT:

```
------------------------------------------------------------------
 [<label>] XS [<cond>] : <CP/M file name> [<optional parameters>]
------------------------------------------------------------------
```

DESCRIPTION: Allows exit from PILOT with overlay and execution of a CP/M command file. The information to the right of the colon can contain any legal input to the CP/M console command processor.

EXAMPLE:

```
------------------------------------------------------------------
     T: Now that you understand what to do, we will use DDT to
      :  look inside the test program.
     XS: DDT TESTPROG.COM
------------------------------------------------------------------
```

4.5 SET PARAMETERS

The statements described in this section specify the way in which PILOT will accept input and present output.

STATEMENT: SET PARAMETERS (SET:)

FORMAT:

```
-----------------------------------------------------------------
                    [<label>] SET [<cond>] : S=n
                    [<label>] SET [<cond>] : M=n
                    [<label>] SET [<cond>] : P=n
                    SET S=n (etc.)   (at PILOT restart)
-----------------------------------------------------------------
```

DESCRIPTION: Sets the parameters of display speed, memory setting, or print echo either during the execution of a PILOT program or as a direct action at PILOT restart.

SET S=n sets display speed of the screen, where n determines the speed with 0 fastest and 9 slowest. You don't have to set this parameter unless you are dissatisfied with the default display speed.

SET M=n determines the upper limit of the program buffer. The greater the value of M, the larger your programs and text files can be. When PILOT is initialized, the buffer is as large as it can be. You need to use SET only if you want to make the buffer smaller.

Any program or text that is stored in the program buffer at the time of this command is lost, so it is a good idea to change this parameter either before the intended file is put into the buffer, or after it has been saved.

M must be a decimal number that points to a mamory location LOWER than the lowest memory location used by the disk operating system in use. Remember that the number you specify in the PILOT SET command must be in base 10.

The greatest value possible for M is 32767 Decimal, or 7FFF Hexadecimal. If you want to set an upper bound higher than 7FFF, use the following formula: to set a limit of n, assign to M a value of -(65536-n). Thus, to set an upper bound of 33023 (80FF), enter SET:M=-32511.

SET P=n allows an echo of screen display to be sent to the CP/M list device. It will be normally used to collect a print of user interaction. SET P=1 turns printing on, SET P=0 turns it off.

EXAMPLE:

```
-----------------------------------------------------------------
     SET: S=5
     SET: P=1
-----------------------------------------------------------------
```

STATEMENT: INPUT MAXIMUM NUMBER OF CHARACTERS (INMAX:)

FORMAT:

```
[<label>] INMAX [<cond>] : <integer>
[<label>] INMAX [<cond>] : <numeric variable>
```

DESCRIPTION: Limits the number of characters to be considered by subsequent A statements. The limit must be expressed as either an integer (1-80) or a numeric variable name, and must appear immediately to the right of the colon. (The maximum number of characters includes blanks; e.g., the strings "123 123" and "1231231" are both seven characters long. The maximum number does NOT include the carriage return.)

If the user supplies the maximum number of characters, PILOT will add its own carriage return, and regard the response given as complete. Any additional text that the user types will be disregarded. (It will not even be visible on the screen.) If INMAX is set to 1, for instance, PILOT will react immediately to the first character entered, without the user's having to press the carriage return key. Of course, if the user gives a carriage return before the character limit is reached, the response is regarded as complete at that point.

LIMITATIONS: INMAX values should be set between 1 and 80. If you do not set INMAX, PILOT allows a length corresponding to the width of the video screen. A greater value causes a harmless overflow to the next line on the screen. INMAX may be set and altered any number of times during a PILOT program.

The entry of an immediate command will temporarily override INMAX; upon return to program execution, the former value of INMAX will be restored.

EXAMPLE:

```
(PILOT program)                     (display)

T:Enter a five digit number.     Enter a five digit number.
  INMAX:5                        (User enters 123456.)
A:#N                             12345
T:Thank you.                     Thank you.
T:You entered #N.                You entered 12345.
```

4.6 FILE MANIPULATION INSTRUCTIONS

The statements described in this section let you collect, store, and retrieve information in files. (The definition and a few of the characteristics of a file are discussed in Section 1.4.) In general, you will be writing files to serve the following purposes:

1) to save a program or data that you have just entered or altered using EDIT, or

2) to make a record of data accumulated during the execution of a PILOT program.

Every time you LOAD or GET a program from the disk, you are reading a file.

In the case of CP/M, program files are in a standard CP/M format, with file type "PLT". Before you use a brand new disk, you will have to "format" the disk using a CP/M utility program.

Every file must have a name, and no two files on the same diskette may have the same name. A file name consists of 1 to 8 characters. In CP/M, the following characters must be avoided:

space	
less than	<
greater than	>
period	.
comma	,
semicolon	;
colon	:
equals	=
question mark	?
asterisk	*
left bracket	[
right bracket	]

If there is more than one disk in your system, there may be a prefix to indicate which disk drive is being referenced.

```
B:SAM
```

denotes a file called SAM on the disk in unit B.

STATEMENT: CREATE DATA FILE (CREATEF:)
KILL DATA FILE (KILLF:)

FORMAT:

```
[<label>] CREATEF [<cond>] : <name>
[<label>] KILLF   [<cond>] : <name>
```

DESCRIPTION: If a file is to be used for collecting data, it must first be created on a disk; if a file is to be discarded altogether, it must be killed.

When you create a file with CREATEF, you do not actually start collecting data in the file. Rather, you tell the system that the file will exist, and what the name of the file will be. Thus, the command must include a filename; if the file is to be created on a disk other than the default unit, a unit designation must be affixed to the name.

No two files on the same disk may have the same name. On the other hand, it is possible for two files on different disks to have the same name. Unless the files contain identical data--that is, unless one of the files is your backup copy for the other-- it is not normally wise to give two files the same name.

The KILLF command has the function opposite to that of CREATEF. Once a file has been killed, the file and the information in it are lost.

The KILLF command requires that the full file name be given. Thus, a PILOT program saved with "SAVE MYPROG" requires the command "KILLF MYPROG.PLT" in order to remove it.

These statements may be used as immediate commands at PILOT restart with or without the final "F".

EXAMPLE:

```
CREATEF:B:MATHTST1  (creates a file called MATHTST1
                     on the disk in unit B)
KILLF:B:MATHTST     (causes B:MATHTST1 and all information
                     in it to be lost)
```

STATEMENT: OPEN DATA FILE (OPENF:)
CLOSE DATA FILE (CLOSEF:)

FORMAT:

```
-----------------------------------------------------------------
     [<label>] OPENF  [<cond>] : <name>,<numeric variable>
     [<label>] CLOSEF [<cond>] : <expression>
-----------------------------------------------------------------
```

DESCRIPTION: To store data in a file, it is necessary to "open" the file; when data collection has been completed, the file should be "closed."

Opening a file is not the same as creating it. When you CREATE a file, you simply establish its existence on the disk. When you OPEN a file, you open a line of communication between the existing file and your PILOT program. The OPENF statement must occur before the first WRITE, RW, APPEND, REWIND, or EOF statement that refers to that file in the PILOT program. (If you open more than one file for data collection, you must open each file before the first program statement that refers to it).

As a result of the OPENF statement, the numeric variable included on the statement line will be assigned a value. That value is the file number associated with the file. All subsequent WR, RW, APPEND, REWIND, and EOF statements that manipulate the file must refer to it not by its file name, but by a numerical variable equal to the file number (see the format of those statements). If you always use the original variable name to refer to the file, you will not need to be concerned with the file number itself. For example, if you write

```
OPENF: B:MATHTST1,M
```

every subsequent statement referring to the file can be of the form

```
<statement>: M
```

Once a file has been opened, data will be written into it starting at the beginning of the file; any data already existing in the file will be overwritten. Beware of accidentally destroying one of your existing files by OPENing it and writing on it for a second time!

If the file that you are trying to open does not yet exist on the disk, it will be created automatically.

The CLOSEF statement closes the line of communication between the program and the indicated file, ensuring that all data from prior WRITE statements are actually recorded on the disk. PILOT will automatically close any file that is open at the end of a program; nonetheless it is a good idea to close an open file after the last statement that refers to it. There is a limit on the number of files that can be open at once. The current limit is two. If you try to open too many files, an error message will be displayed. If you press RETURN, the program will continue to run, but the file will not have been opened. To solve the problem that caused the error message, stop the program and use the EDITor to alter the offending code.

EXAMPLE:

```
-----------------------------------------------------------------
(PILOT program)

     OPENF:B:FRED,F      (file "FRED" is opened on unit B;
                          a file number is assigned to F)
     WR: F               (user's last response is recorded in
                          the file named FRED, denoted by the
                          variable containing its file number)
     CLOSEF: F           (FRED is closed)
     E:
-----------------------------------------------------------------
```

STATEMEMT: WRITE INTO DATA FILE (WRITE: or WR:)

FORMAT:

```
              [<label>] WRITE  [<cond>] : <expression>
              [<label>] WR     [<cond>] : <expression>
```

DESCRIPTION: Each of these statements causes the last entry to an Accept statement to be written into the file whose number is indicated by the expression on the statement line. (The expression may be the numeric variable that represents the file number.)

A WRITE statement will be executed only if the specified file is open: that is, the file must have been opened by an OPENF statement earlier in the program, and it may not yet have been closed by a CLOSEF statement. Successive WRITE statements will add to the contents of the file until the file is closed with CLOSEF. If the file is opened again, WRITE statements will then replace the previous contents, which will be lost.

If you write a sequence like

```
     T: How old are you?
     A: #A
     ....
     ....
     C: A=A+1
     ....
     ....
     WR: F
```

the value written to the file will be the number assigned to A by the Accept statement, not the new value of A formed by the C statement. If you wanted to record the current value of A, you should use the RW statement, described next.

EXAMPLE:

(PILOT program)

```
     OPENF: CITY,C      ("CITY" is opened on the disk in unit 0)
     T: NAME A CITY
     A:                 (user types "Pittsburg")
     WR: C              (Pittsburg is written into the file
                         called CITY on unit 0)
```

STATEMENT: REMARK WRITE (RW:)

FORMAT:

```
             [<label>] RW [<cond>] : <expression>,<text>
```

DESCRIPTION: This statement causes the text to be written into the file whose number is indicated by the expression on the statement line. The expression may be the numeric variable that represents the file number.

The text may contain the names of string or numeric variables. When the text is written to the output file, each variable is replaced by its current value. In this respect, a RW statement is just like a T statement, except that its destination is a file rather than display on the screen.

A RW statement will be executed only if the specified file is open: that is, the file must have been opened by an OPENF statement earlier in the program, and it may not yet have been closed by a CLOSEF statement.

EXAMPLE:

```
-----------------------------------------------------------------
(Pilot program)

   OPENF: PERSONAL,P
   T: How many brothers do you have?
    : (Type 0 if none.)
   A:#B                         (User types "1")
   T: How many brothers and sisters do you have altogether?
    : (Type 0 if none.)
   A:#S                         (User types "4")
     R: Calculate number of sisters.
   C: S=S-B
   RW: P, YOU HAVE #S SISTERS  (The text "YOU HAVE 3 SISTERS"
                                is written into the file called
                                PERSONAL on unit 0.)
-----------------------------------------------------------------
```

STATEMENT: APPEND TO DATA FILE (APPEND:)

FORMAT:

```
-----------------------------------------------------------------------
                [<label>] APPEND [<cond>] : <expression>
-----------------------------------------------------------------------
```

DESCRIPTION: The APPEND statement allows information to be added to the contents of a file that has been closed and reopened.

The normal operation of the first WRITE statement after OPENF is to overwrite information present in an existing file. The APPEND statement may be used after OPENF (once) to retain any existing information in the file, moving the point at which WRITE will add information to the current end-of-file location.

EXAMPLE:

```
-----------------------------------------------------------------------
(PILOT program)

     CLOSEF: C            (Prior use of "CITY" file using file
     ....                      number C has been closed)
     ....
     ....

     OPENF: CITY,A        ("CITY" opened again, using file no. A)
     APPEND: A            (retains old data, prepares to add new)
     T: Name an additional city.
     A:
     WR: A
-----------------------------------------------------------------------
```

STATEMENT: READ FILE (READ:)

FORMAT:

```
--------------------------------------------------------------------
     [<label>] READ [<cond>] : <expression>[,<string variable>]
--------------------------------------------------------------------
```

DESCRIPTION: Allows reading a line of data from an existing disk file into the entry buffer and optionally into a string variable. The file must be opened using OPENF and a file number or variable assigned. This is the value referenced by <expression>.

The optional addition of a string variable name will cause the data to be also stored in memory under that name.

EXAMPLE:

```
--------------------------------------------------------------------
(PILOT program)

    OPENF: B:DATAFILE.TXT,X (opens file for reading or writing)
    READ: X                 (one line from file read)
    M:SMITH                 (see if "SMITH" is in the line)
    UY: *FOUND
--------------------------------------------------------------------
```

STATEMENT: REWIND FILE (REWIND:)

FORMAT:

```
----------------------------------------------------------------
              [<label>] REWIND [<cond>] : <expression>
----------------------------------------------------------------
```

DESCRIPTION: This statement causes the specified file (i.e., the file whose number is signified by the expression) to be "rewound." If one imagines a pointer that indicates where in the file the next item of data will be written, the REWIND statement positions the pointer to the beginning of the file. Subsequent WRITE and RW statements will therefore write over any data already in the file.

When you open a file with the OPENF statement, the file is, in effect, rewound. Any data that your program collects will be written over the data already existing in the file.

A REWIND statement will be executed only if the specified file is open: that is, the file must have been opened by an OPENF statement earlier in the program, and it may not yet have been closed by a CLOSEF statement.

EXAMPLE:

```
----------------------------------------------------------------
   REWIND: F     (rewinds the file represented by F
----------------------------------------------------------------
```

STATEMENT: END OF FILE (EOF:)

FORMAT:

```
-----------------------------------------------------------------
               [<label>] EOF [<cond>] : <expression>
-----------------------------------------------------------------
```

DESCRIPTION: This statement ends the file at the current position, i.e., any data following the current position will be discarded. It is advisable to use this statement before closing an existing file that has been rewritten or a file that has been rewound and had new data written into it. In fact, it is good practice to insert an EOF statement before every CLOSEF statement in your PILOT program, just to be sure that no extraneous data are left at the end of the file.

If you end a file but do not close it, a subsequent RW or WR statement will cause data to be added to the end of the file.

EXAMPLE:

```
-----------------------------------------------------------------
(PILOT program)

    ....
    ....
    REWIND: N      (rewinds the file represented by N)
    ....
    ....
    EOF: N         (ends the file represented by N)
-----------------------------------------------------------------
```

STATEMENT: LOAD NEW PROGRAM OR PROGRAM SEGMENT (LOAD:)

FORMAT:

```
------------------------------------------------------------------
          [<label>] LOAD [<cond>] : <name>
          LOAD <name>                         (at PILOT restart)
          XEQ  <name>  is an allowed synonym
------------------------------------------------------------------
```

DESCRIPTION: This statement loads the named file into the program buffer and initiates execution if the file contains a program. If the file you are attempting to load is not a PILOT program, it is simply displayed on the screen.

The specified name must be that of an existing file. If the unit designation is omitted, the file will be presumed to reside on the default unit disk. The file type ".PLT" will be supplied if an alternative is not given.

If you want to load a file but not to execute it - for example, if you want to EDIT or LIST it - use the GET command, rather than LOAD. GET is described in Section 4.7.

LOAD may be used in the immediate mode to load a new program and also in a program sequence to replace the current segment of a long program with the next segment.

Any time that you return to PILOT restart to LOAD a new segment of a large program, you have the option of using the NEW:# command to set all numeric variables to zero; otherwise, the values for these variables will be retained. In the case of a program segment larger than the previous one, there is a good chance that LOADing the larger segment will cause the loss of some string variables. A way to avoid this difficulty is to arrange your program segments so that each is shorter than the last. (A program segment is just a program that is intended to run in sequence with other programs.) The INFO command, which is described in Section 4.7, will enable you to determine the size of any program or program segment.

EXAMPLE:

```
------------------------------------------------------------------
(PILOT program)

        .....                  .....
        .....                  .....
        LOAD: NEWSEG        (causes NEWSEG to be loaded from the
                             default disk)
------------------------------------------------------------------
```

4.7 COMMANDS USUALLY USED IN IMMEDIATE MODE

Although all of these commands except NEW can be entered as program statements, they will probably be most useful from PILOT restart. After a command is executed from the restart point, PILOT expects to receive a carriage return. When you type the carriage return, you will be back at restart and from there can enter another command or return to program execution.

COMMAND: EXIT FROM PILOT (BYE:)

FORMAT:

```
------------------------------------------------------------------
                 [<label>] BYE [<cond>] :
------------------------------------------------------------------
```

DESCRIPTION: Causes PILOT to discontinue operation, and returns control to the disk operating system. Once control has returned to the operating system (CP/M), you can

1) start PILOT again by typing PILOT <cr>, or restart PILOT by initiating execution at location 103H.

2) Run some other program under the operating system, or

3) remove your disk and turn off the system.

This instruction is normally used in the immediate mode at the PILOT restart point.

EXAMPLE:

```
------------------------------------------------------------------
                 BYE         (at PILOT restart)
------------------------------------------------------------------
```

COMMAND: GET PROGRAM (GET:)

FORMAT:

```
--------------------------------------------------------------------
     GET  <file name>                      (at PILOT restart)
--------------------------------------------------------------------
```

DESCRIPTION: Loads the named program from disk into the program buffer, without initiating execution.

This command provides a convenient way of loading a file into memory for a purpose other than that of running the program: for example, to EDIT or LIST. Any file created in PILOT may be loaded in this way. Once the file has been loaded, PILOT returns to restart. If you decide you want to execute the program after all, type RUN.

The specified name must be that of an existing file. If the unit designation is omitted, the file will be presumed to reside on the default disk. The file type ".PLT" will be supplied if an alternative is not given

EXAMPLE:

```
--------------------------------------------------------------------
    (command form)

      GET B:BOX    (loads BOX from the disk in unit B)

--------------------------------------------------------------------
```

COMMAND: RUN PROGRAM (RUN:)

FORMAT:

```
RUN            (at PILOT restart)
```

DESCRIPTION: Starts execution of the program in the program buffer. If there is no program in the program buffer, nothing happens.

If you have loaded a file using GET, or if you have discontinued the execution of a program, you can give this command from restart to execute the program, beginning at its first executable statement, i.e., not wherever you left off.

EXAMPLE:

(command form)	(function)
RUN	Starts execution of the program in memory

COMMAND: LIST PROGRAM (LIST:)

FORMAT:

```
-----------------------------------------------------------------
                [<label>] LIST [<cond>] :
                LIST                          (at PILOT restart)
                LIST:P                        (at PILOT restart)
-----------------------------------------------------------------
```

DESCRIPTION: Lists the current contents of the program buffer on the display screen or on a printer.

The list command may be given from within a program or from PILOT restart. If you want to stop a LISTing that is in progress, hit the ESCAPE key on your keyboard. Control will return to the program in memory or to PILOT restart, depending on the point from which the command was given. If you want to halt a LISTing momentarily, you can hit the space-bar. The LISTing resumes when another key is pressed. If you are LISTing on the screen, you can reset the video display speed by hitting a number key: 0 is fastest, 9 is slowest.

KEYS 0-9: Reset video display speed; 0=fast, 9=slow.

EXAMPLES:

```
-----------------------------------------------------------------
        (command form)           (function)

            LIST       lists current program on the console
            LIST:P     lists current program on a printer
                       via the CP/M list device.
-----------------------------------------------------------------
```

COMMAND: SAVE PROGRAM (SAVE)

FORMAT:

```
--------------------------------------------------------------
               SAVE <file name>        (at PILOT restart)
--------------------------------------------------------------
```

DESCRIPTION: SAVE writes onto disk the PILOT program or other file currently in the buffer. SAVEing a file is the only way to record it after it has been EDITed.

If the file already exists on disk at the time of the command, the program in memory is simply put into that file, supplanting any previous contents. If the file does not yet exist at the time of the command, it is created, and the program in memory is put into it.

A filename must be specified as part of the SAVE command.

The file type ".PLT" will be supplied if an alternative is not given. If a unit designation is omitted the file will be saved on the default unit disk.

EXAMPLE:

(command form)	(function)
SAVE TEST1	Saves a file called "TEST1" on the default disk drive

COMMAND: ERASE CURRENT PROGRAM (NEW)

FORMAT:

```
-----------------------------------------------------------------
                 NEW      ( at PILOT  restart)
-----------------------------------------------------------------
```

DESCRIPTION: Deletes the current program from memory, along with all string variables. If there is no program in the program buffer, nothing happens. NEW can be entered only from PILOT restart.

The PILOT interpreter is not deleted by this command.

COMMAND: FILE SIZE INFORMATION (INFO:)

FORMAT:

```
-----------------------------------------------------------------
             INFO             (at PILOT restart)
-----------------------------------------------------------------
```

DESCRIPTION: Tells where your PILOT program is in the memory of the computer. Four items of information will be displayed on the screen:

1) MEMORY SETTING - This number, which is in base 16 (10 through 15 are denoted by the letters A through F), tells where the program buffer ends.
2) FILE BEGINNING - This number, also in base 16, indicates where the program buffer starts.
3) CHARACTERS IN FILE, including blanks - This number, in base 10, indicates the number of characters in the program.
4) SPACE REMAINING - This number, also in base 10, indicates the number of unused characters remaining in memory.

The INFO command helps you to decide whether your program is too long relative to the amount of memory allocated to it. (Remember that string variables are stored along with the program in the program buffer.) If your program almost fills the buffer, you should consider shortening the program, dividing it into segments, or assigning more memory (with the SET M= statement.)

EXAMPLE:

```
-----------------------------------------------------------------
       (command form)               (display)

            INFO                 5FFFH Memory setting.
                                 1D69H File beginning.
                                 2142 Characters in file.
                                 14905 Space remaining
-----------------------------------------------------------------
```

COMMAND: DIRECTORY LIST (DIR:)

FORMAT:

```
-----------------------------------------------------------------
            DIR [<unit:>]     (at PILOT restart)
-----------------------------------------------------------------
```

DESCRIPTION: Displays files on default disk unit or designated disk unit.

When PILOT is configured for your specific terminal, you are given a choice of whether DIR should display all files on the designated unit or display only files of type .PLT .

The listing of files is similar to the DIR command in CP/M, but it is convenient to be able to obtain this information without leaving PILOT.

4.8 CONTROL OF VIDEO CASSETTE RECORDER

PILOT provides statements which can control random access to video playback, both picture and sound. The computer must be connected to specific video cassette equipment via a parallel port. If the computer uses memory-mapped display, the video picture can be displayed on the same monitor screen used for normal PILOT interaction. If a separate CRT is used, then this is not easily arranged, and it will likely be necessary to present the video image from the cassette player via a separate monitor.

Sony models which currently allow this operation are:

Model SLO-320.	A recorder/player in 1/2 in. Betamax format.
Model SLO-323.	Newer model of the above.
Model VP-2011.	A player only in 3/4 in. U-matic format.

Models SLO-320 and SLO-323 allow the video picture to be displayed on the same monitor screen used for normal PILOT interaction. A custom connector cord is required between a parallel port connection on the computer and the cassette player. No changes are required to either the computer or the cassette player.

STATEMENT	START VIDEO COUNTER	(VSTART:)
	SET VIDEO COUNTER	(VSET:)
	VIEW SELECTED VIDEO	(VIEW:)

FORMAT:

```
-----------------------------------------------------------------
      [<label>] VSTART [<cond>]:
      [<label>] VSET   [<cond>]: <count>
      [<label>] VIEW   [<cond>]: [<start#>] [,<stop#>]
-----------------------------------------------------------------
```

DESCRIPTION: A search for a specific place at which video playback is to start or stop is dependent on a count of pulses recorded on the cassette. The first step is to rewind the cassette to its starting point and to reset the counter in PILOT which keeps track of the pulses. The VSTART statement will rewind the cassette and set the counter to zero. If for some reason you do not wish to rewind the cassette and you know what the count should be, the VSET statement can be used to set the count to a specific number. The count used in PILOT approximates seconds of elapsed time, and searches should be accurate to within a second or two.

The VIEW statement is used to initiate search and viewing of a particular segment of recorded video. The start number and stop number approximate seconds of time relative to the zero point set by VSTART or by VSET. The start and stop numbers may be numeric constants or variables. If the start number is missing, then viewing begins from the present location. If the stop number is missing, then the cassette player will continue to play. Control can be returned to the next PILOT statement by pressing ESCAPE.

EXAMPLE:

```
---------------------------------------------------------------
     VSTART:        (rewinds cassette and sets counter to zero)
     T: Push RETURN to see a demonstration of how these parts
     T: can be put together.
     A:
     VIEW: 420,505
     T: Now try to answer the question again.
---------------------------------------------------------------
```

Directions for creating the necessary connection cord:
This cord connects between a parallel port on the computer and the control socket on the Sony cassette unit.

1. Obtain Sony part No. 1-551-317-00 which is a replacement cord for RM-300 random search control.

2. Open the shell of the special connector and move the pin attached to the orange wire from position 9 to position 5.

3. Remove and discard the small Molex connector from the other end of the cord, shortening all wires to 1 in. from the point where they exit from the outer insulation. Leave the grey and light green wires connected together, cutting them 1/4 in. past where they join.

4. Strip 1/4 in. of insulation from the end of all wires except white, brown, and copper braid (covered with black spagetti) Tin the stripped wires. The white, brown, and copper wires are not used.

5. Connect as follows to a 25 pin-D subminiature female connector:

pin	wire	parallel port signal
2	grey & light green	signal ground
6	light blue	input data, bit 7
7	black	input data, bit 6
8	yellow	input data, bit 5
18	dark green	output data, bit 7
19	dark blue	output data, bit 6
20	violet	output data, bit 5
21	red	output data, bit 4
22	orange	output data, bit 3

The CP/M BIOS section should direct RDR: and PUN: logical devices (input and output respectively) to a parallel port used for control of the cassette unit. PILOT will receive and send appropriate control bits as shown in the table on page 4-48 in response to VSTART, VSET, and VIEW commands.

Timing is based on a computer running at 2mh. For faster computers, the value of PATIM in the system data area will have to be reduced by a patch using DDT. One of the files on the PILOT distribution disk is NVPILOT.PRN which is an assembly listing of the system data area. This provides the address where the current value if PATIM can be changed.

In order to view computer-generated text and video playback on a memory-mapped video screen, connect video output from the computer to Video In on the cassette player, and Video Out on the cassette player to the monitor. This will require a cord with a UHF connector on the computer end and a BNC connector on the Sony cassette recorder end.

SECTION 5

ENTERING AND EDITING YOUR PILOT PROGRAM

5.1 THE EDIT COMMAND (EDIT)

FORMAT:

```
-----------------------------------------------------------------
     EDIT                    (At PILOT restart)
     \EDIT:                  (At any A statement)
-----------------------------------------------------------------
```

DESCRIPTION: Makes available a new set of commands that can be used to create and alter text and program files. In EDIT the cursor may be positioned anywhere on the screen, lines may be scrolled up and down, and characters and entire lines may be inserted or deleted. There are also provisions for searching the file for strings, and for moving quickly to any one-tenth portion of the file from 0 to 9.

EDIT is different from all other PILOT commands, in that when you enter it, or execute it within a program, you move into an operational mode that is really quite separate from the interpreter. The EDITor is a subsystem of PILOT. It enables you to make changes in any text file (whether or not it was created by PILOT, or for use with PILOT.) The EDITor does not not know or care whether you are working with a PILOT program, a BASIC program, a list of student responses, or a letter to your doctor.

The EDIT command may be entered from immediate mode by typing "EDIT". If you want to enter a new PILOT program, give the EDIT command from PILOT restart, without LOADing or READing anything into the program buffer. If you want to EDIT a file that you have created in PILOT with WRite statements, use the GET command to load it into memory without executing it.

When you enter the EDIT command from immediate mode, the first page of the file in memory is displayed, with the cursor at line one and position one (column 0). If you give the command in response to an A statement, or if EDIT: is actually a statement in your PILOT program, the first line on the display will be the A statement or the EDIT statement, respectively. If the existing text is not enough to fill the screen, the remaining portion of the screen will be filled with blanks. If you are entering a file for the first time, just begin typing; otherwise, the file displayed on the screen is ready for EDITing. The next few pages tell how to go about changing a file by using control characters.

The EDIT command may also be entered, with a backward slash preceding and a colon following ("\EDIT:"), in response to any A statement. Under these conditions, the A statement at which you entered the command will be at the top of the screen and any following text will fill the rest of the screen.

Remember, the control key along with another key produces a control character. Press CTRL-F to exit from the EDITor and return control to command mode. If you want to execute the program you have just EDITed, type RUN.

Below is a list of the command keys used by the EDITor. A more complete description of each command is given after the list. These are default commands that will be used unless modified during a configuration process. Other control codes or special character sequences may be substituted, thus allowing use of special keys on some terminals. The specific memory locations to be changed will be found in the file NVPILOT.PRN.

5.2 COMMAND KEY LIST

CONTROL KEYS

CONTROL W - move cursor up one line
CONTROL Z - move cursor down one line
CONTROL A - move cursor left one character
CONTROL S - move cursor right one character
CONTROL T - toggle insert character mode; ON/OFF
CONTROL G - delete character under cursor
CONTROL B - insert line above cursor
CONTROL P - delete line
CONTROL N - move cursor to upper left corner of screen
CONTROL E - move file up one line
CONTROL X - move file down one line
CONTROL R - scroll file up one-half page
CONTROL C - scroll file down one-half page
CONTROL Q - move cursor to a mid line, column 1
CONTROL V - initiate string search mode
CONTROL L - continue search for string
CONTROL O - cause reverse video for subsequent characters
CONTROL F - exit from the editor or editor text search
TAB - move cursor to the next tab position (CONTROL-I)
RETURN - insert line below cursor (CONTROL-M)
LINE FEED - delete all text from the right of the cursor and position cursor one line down (CONTROL-J)
BACKSPACE - backspace and erase a character (CONTROL-H)
DELETE - (or RUBOUT) same action as backspace
REPEAT - re-enter whatever other key, or combination of keys, is depressed; continue while REPEAT key is held

When you leave EDIT mode by pressing ctrl-F, the program you have prepared resides in memory and is ready to RUN. But it has not been saved. If you wish it to be stored in a disk file for later use, you must type "SAVE <file name>" before leaving PILOT.

5.3 DETAILED COMMAND DESCRIPTION

5.3.1 Cursor Positioning Commands

NOTE: Moving the cursor does not change the text.

The keys A,S,W,Z form a diamond on the imput keyboard. When pressed simultaneously with the 'CTRL' (control) key, they move the cursor as indicated below:

CONTROL W	move cursor up one line
CONTROL Z	move cursor down one line
CONTROL A	move cursor left one character
CONTROL S	move cursor right one character
CONTROL Q	move cursor to mid screen, column 1

5.3.2 Screen Scroll Commands

Screen scroll commands are provided to allow the file to be "rolled" through the screen area until the desired file line is reached.

CONTROL E	scroll up one line
CONTROL X	scroll down one line
CONTROL R	scroll up one-half page
CONTROL C	scroll down one-half page

5.3.3 Direct File Positioning Commands

In addition to cursor positioning controls, the EDITor offers a way of searching for a specific string of text within your file. The search command is CONTROL V.

CONTROL V	editor text search

When you type control V, the last line of the display is cleared and the normal video reversed for this line. If an extra line is available, such as a 25th line on some terminals, this line is activated and used. The cursor is placed at the first position in the line. At this point the EDITor is waiting for you to enter either: 1) An input line consisting of one or more characters, or 2) a single digit. The input is terminated by a carriage return.

1. Character entry:

Any occurrence of the string entered, regardless of preceding or following characters, will represent a find. Therefore, only enough characters to define the desired text uniquely need be supplied. As an example, "the qu" can be used to locate a line in the file containing "the quick brown fox."

Upon receiving a carriage return, the EDITor searches the file, beginning one line below the current cursor position, until a string match is made or until the end of the file is reached. At the beginning of a file, the search begins at the first line. If a match is found, the EDITor positions the line containing the match at the top of the screen.

CONTROL L continue search

If you wish to continue searching for text matches after having left edit text search, pressing CONTROL L will cause continued searching for the string that was last designated. The EDITor resumes the search at the first line following that in which the cursor resides at the time of the command and continues until a match is made or until the end of file is reached. This command may be given as often as is desired.

2. Digit entry:

If you enter a digit from 0 to 9 in the command line, the file will be scrolled so that the top line on the video display screen marks the end of that tenth of the file which corresponds to the number entered. Thus, if the number is 5, the file will be positioned at the half-way point. If 0 is entered, the file will be positioned at the beginning. ESCAPE will cause an exit from editor text search and a return to EDIT mode.

5.3.4 File Modification Commands

CONTROL T character insert mode switch (on-off-on....)

Normal file characters input from the terminal are placed in the file in either of two modes. These modes, normal and insert, are alternately selected using the insert mode control.

When insert mode is OFF (default mode when EDITor is entered), each character that you type replaces what was formerly at the current cursor position, and the cursor moves to the right one place. When insert mode is ON, characters are actually inserted BEFORE the current cursor position, moving the character at that location, and any characters to the right of it, one position to the right. The cursor also advances one position. A line contains a maximum number of characters, so you may begin to lose text that is pushed off the screen by the insertion.

CONTROL G delete character

The delete character command removes the character at the current cursor position and moves each character to the right of the cursor one position to the left.

CONTROL B insert line command

The insert line command puts a new blank line at the present cursor position, and moves each subsequent line of the file one row down. The cursor is moved to the first character position of the new line. Use this command to insert a new line "above" the current line.

CONTROL P delete line command

This control removes the current cursor line from the file.

LINEFEED (CONTROL J) blank remaining line

Linefeed deletes all characters to the right of the current cursor position (on the cursor line). The cursor appears at the beginning of the next line in the file.

CARRIAGE RETURN (CONTROL M) scroll up & insert line

Carriage return scrolls up one line and inserts a blank line in the file. The cursor is moved to the first character position of the new line. Use RETURN to insert a line 'BELOW' the current cursor position. No characters on the current line are deleted. The exception to this rule is that, if a file contains fewer than one page of text, RETURN will open a new blank line below the last line of text but will not scroll the file.

TAB (CONTROL I) horizontal tab

When TAB is pressed, the cursor will move to the next column divisible by eight (columns 8, 16, 32, ...).

CONTROL O reverse video mode (on-off-on....)

When reverse video mode is off, characters are displayed white on a black background. This control causes subsequent characters to be reversed as black on white until another CTRL-O is entered.

SECTION 6

ERROR MESSAGES AND HOW TO DEAL WITH THEM

Most of these error messages have been introduced at other points in this manual. Here we present a summary which you may find useful as a reference. In each case, an example of the message, as it might appear to a user, is followed by a brief explanation of its likely implications.

```
<program text>
*UNRECOGNIZED PILOT STATEMENT
```

The text displayed above is not recognized as a valid PILOT statement. Check the format of what you see displayed. Perhaps a colon is missing. Type "\EDIT:" to enter the EDITor to fix the problem.

```
*<label> or PR: or M: -NOT FOUND
```

The PILOT program in memory specifies a jump to a statement or a subroutine that does not exist. Check for a possible misspelling of the label in the J or U statement. The expression enclosed by the angle brackets is the label as it appears spelled in that statement. If the item not found is a PR or M statement, there is probably no such statement between the J or U statement and the end of the program.

```
<C statement>
*CANNOT EVALUATE THE EXPRESSION
```

A compute statement has incorrect format. Maybe it specifies an operation that is not allowed, e.g. C: R= A OR B is incorrect, since "OR" is not a recognized operation.

```
<C statement>
*VALUE OUT OF RANGE(-32768 TO 32767)
```

The compute statement in the angle brackets has either an element or a result that is not in the possible range of numbers allowed in PILOT. In such a case, the value assigned to the designated variable is -32768 for a negative number or 32767 for a positive number.

```
<string variable name>
```

If you refer to a string variable to which no value has been assigned, and your immediate intention is not to accept a value for that variable--say, for example, you wanted to print or write it--that variable name, itself, is displayed. You may want to make such a reference intentionally. If not, check for a spelling error, make sure that the variable you are naming is the one that you really intended to name, or find out if your statement referring to a given variable really follows the assignment of a value to that variable. Did you put a VNEW:$ statement in your program and then try to retrieve a value that you had erased?

*NUMERIC RESPONSE REQUIRED

The user of a PILOT program receives this message if he gives a non-numeric reply to an A statement that requests a value for a numeric variable. The solution is generally to give a numeric reply.

*USE DEPTH EXCEEDED

The program being executed has more than seven subroutines in execution. See discussion of this for the USE: statement, p. 28.

*NO ROOM

There is insufficient space in memory for the string variable that you are trying to store. You may want to use a VNEW:$ statement to clear the area occupied by variables that you are no longer using.

The following error messages may occur during file manipulations and result from an error condition generated by CP/M. Unless the error indicates a machine malfunction, you may press the RETURN key and your program will continue at the next statement. It is a good idea, however, to stop your program and investigate the reason that an error message was generated.

```
ERROR WHILE CLOSING FILE
ERROR WHILE CREATING FILE
DRIVE NUMBER OUT OF RANGE
UNAVAILABLE CP/M FUNCTION CALLED
FILE NUMBER OUT OF RANGE
LIMIT FILE NAME TO 8 CHARACTERS
FILE NOT OPEN
FILE NOT FOUND
TOO MANY FILES OPEN
END OF FILE
```

If you have not received a specific error message, but something seems to be going wrong, consider the following possibilities:

1) You have used more than one label with the same name; control has been passed to the first such label to occur in the program, rather than to the label to which you intended to jump. Rename one of the labels.

2) You have loaded a program that is too big or a text file that is not executable as a program.

3) You have attempted to enter a response longer than the present value of INMAX will allow. (This problem does not apply to an immediate command. Immediate commands override INMAX.)

4) You have misspelled or mispunctuated something.

5) You have LOADed or READ something into memory and obliterated the former program (and possibly some of your string variables.)

6) You have entered something that is not a command, and it has simply been displayed.

7) You have neglected to mark the end of a subroutine with an End statement.

8) You have relied on a zero value for a numeric variable, but you have not re-initialized numeric variables since the last program was executed.

PILOT's built-in Editor allows a program author to make immediate repairs to a faulty program in many cases. When an error message appears, the computer will wait for an entry from the keyboard. If you press RETURN the program will proceed with the next statement, but if you wish to make a change to the program, you may immediately enter EDIT mode by typing "\EDIT:". The last line of the program that has been executed will appear at the top of the screen.

SECTION 7

PROGRAMS TO TEST PILOT FUNCTIONS

7.1 PLTST and related test programs.

The following PILOT programs were developed as tests of PILOT functions and will prove useful as examples in those cases where the description of a particular operation does not seem completely clear in the manual. The programs are provided on the disk so that you can watch PILOT in action. PILOT statements and some PILOT error messages are introduced. The information displayed by the use of T statements is most often an indication of what is happening in the PLTST program itself, so it helps to refer to the program listing when running it.

Please note that these programs use the LOAD statement to transfer control from one to the other, and that they expect the programs to reside on CP/M disk A. If the test programs are on drive B, then the LOAD statements must be modified (e.g. LOAD:B:<file name>) to address the proper disk drive. If all programs are on disk A, then initiate the test by typing:

```
A> PILOT PLTST
```

7.2 WAPP

Another program, WAPP, is provided for both demonstration and test purposes. WAPP stands for "Write a PILOT program," and it asks you to supply one or more multiple-choice questions, with answers and responses keyed to each answer. You don't have to program any of the details, but just answer the prompting questions. WAPP is an example of how one PILOT program can serve as a vehicle for another to be written. You can save, and late execute, the multiple-choice program that WAPP writes for you.

Both PLTST and WAPP are written in PILOT, so there is nothing that either of these programs does that you cannot do with the statements described in this manual. If you cannot figure out how a program does something, either LIST the program or display it in the EDITor, and take a look at the PILOT code.

```
R:      PLTST       TESTS OF PILOT FUNCTIONS
CH:
T:TEST PROGRAM OF PILOT FUNCTIONS
T:
T:YOU CAN TRY PILOT CORE INSTRUCTIONS:
:  T:  TO PRESENT TEXT
:  A:  TO ACCEPT AN ANSWER.
:  M:  TO MATCH ELEMENTS OF AN ANSWER.
:  J:  TO JUMP TO A LABELED PLACE.
:  U:  TO USE A SUBROUTINE.
:  E:  TO END A SUBROUTINE OR THE ENTIRE PROGRAM.
:  C:  FOR LIMITED COMPUTATION.
:  R:  TO INSERT A REMARK WITH NO OPERATIONAL EFFECT.
:
:BEGIN BY TYPING ANYTHING, THEN PRESS 'RETURN'
:  USE DELETE TO ERASE ONE LETTER, AND CTRL/X
:  TO KILL A LINE BEFORE YOU HAVE PRESSED RETURN.
A:$WHAT
   CH:
T:YOU TYPED $WHAT.
T:$WHAT IS WHAT YOU TYPED.
T:   PRESS 'RETURN' TO CONTINUE.
A:
   CH:
T:
:Y OR N APPENDED TO ANY INSTRUCTION MAKES ITS USE CONDITIONAL

:Y: OR N: ALONE ARE SHORTHAND VERSIONS OF TY: OR TN:
T:
*TESTM
T:I WILL LOOK FOR 'ABC', ' DEF', 'GHI ', OR ' JKL '
T:NOTE THE SPACES.  THEY ARE IMPORTANT IF THE LETTERS WE ARE
T:TRYING TO MATCH ARE NOT AT EITHER END OF A LINE.
A:
M:ABC, DEF,GHI , JKL ,
TY:MATCH
TN:NO MATCH
   U:*ASK
   JY:*TESTM
   J:*TESTC

*ASK
T:AGAIN? (Y OR N)
A:
M: Y
E:


*TESTC
   CH:
T:TYPE A NUMBER FROM -32768 TO 32767
   A:#A
T:ANOTHER NUMBER PLEASE
   A:#B
T:YOUR NUMBERS ARE #A AND #B.
```

```
T:I WILL SUBTRACT THEM.
   C:C=A-B
T:#A - #B = #C
T:
T:      TYPE 'NEXT' TO GO ON
A:
M:NEXT
 JN:*TESTC

*TESTPLUS
   CH:
   C:A=0
   C:B=0
T:TESTS CAN BE CONDITIONAL ON A NUMERIC VALUE > 0
T:A NUMERIC VARIABLE IS APPENDED AS IN 'J(X):*LABEL'
T:TYPE 'ABC' OR 'DEF'
A:
M:ABC
 CY:A=1
M:DEF
 CY:B=1
T(A):ABC FOUND
T(B):DEF FOUND
   C:C=A+B
   J(C):PLUSEND
T:NO MATCH OF EITHER ABC OR DEF

*PLUSEND
T:END OF TEST
   U:*ASK
   JY:*TESTPLUS

*MORE
   CH:
T:SOME OTHER INSTRUCTIONS AVAILABLE IN THIS SYSTEM ARE:
:   AS:      TO ACCEPT ONLY ONE CHARACTER (WITH NO 'RETURN').
:   INMAX:   TO LIMIT THE NUMBER OF CHARACTERS ACCEPTED.
:   CH:      TO CLEAR SCREEN AND HOME CURSOR.
:   CL:      TO CLEAR THE REST OF THE CURRENT LINE.
:   CE:      TO CLEAR TO THE END OF THE SCREEN.
:   CA:R,C   TO PLACE THE CURSOR AT POSITION R,C.
:   PA:T     TO PAUSE FOR T SECONDS.
:   FOOT:    TO PROMPT A RESPONSE BEFORE PROCEEDING.
   FOOT:

*TESTIN
   CH:
T:         AS: WILL BE USED TO LOOK FOR 'A'.
T:         PRESS ONLY A SINGLE KEY.  DO NOT PRESS 'RETURN'
T:                  ----                  ---
   AS:
M:A
Y:YOU TYPED 'A'
N:THAT WASN'T 'A'
T:AGAIN? (Y OR N)
```

```
   AS:
   M: Y
 JY:*TESTIN

*TESTCTL
   CH:
   CA:3,10
T:THIS TEXT STARTS AT ROW 3 AND COLUMN 10.
   CA:4,10
T:TYPE ANYTHING UP TO 5 CHARACTERS (HERE WE USE INMAX:5).
   INMAX:5
A:$LIMIT
   INMAX:64
T:YOU TYPED '$LIMIT'.
T:WHEN YOU PRESS 'RETURN' I WILL ERASE THE ABOVE LINE.
A:
   CA:6
   CL:
   CA:8
T:WHEN YOU PRESS 'RETURN' I WILL BLINK THE WORD 'RETURN'
A:
   C:R=8
   C:C=17
   U:*BLINK
   U:*ASK
   JY:*TESTCTL
   J:*PAUSE

*BLINK
   C:X=5
*BL2  CA:R,C
T:
   U:*DELAY
   CA:R,C
T:RETURN
   U:*DELAY
   C:X=X-1
   J(X):*BL2
*DELAY
   C:Y=2
 *DELY
   C:Y=Y-1
   J(Y):*DELY
   E:

*PAUSE
T:WHEN YOU PRESS 'RETURN' THERE WILL BE A 5 SECOND PAUSE.
   A:
   PA:5
   U:*ASK
   JY:*TESTCTL

*ERRORS
   CH:
T:I WILL SHOW SOME ERROR MESSAGES.  AFTER EACH, PRESS 'RETURN'.
```

```
T:
PA:1
T:FOR EXAMPLE,
T:I WILL TRY TO JUMP TO AN UNKNOWN LABEL '*HERE'
T:
   J:*HERE

   CH:
T:THE FOLLOWING C-STATEMENT HAS BAD SYNTAX:
T:   C:X NOT Y
T:
T:THE EXPRESSION FIELD WILL BE DISPLAYED + AN ERROR MESSAGE.
T:
     C:X NOT Y
   CH:
T:A C-STATEMENT CAUSES X TO BECOME GREATER THAN 32767
T:
T:THE STATEMENT CAUSING OVERFLOW WILL BE DISPLAYED.
T:
   C:X=32767+10
T:THE VALUE OF X IS LEFT AT #X
T:A SIMILAR MESSAGE APPEARS WHEN X BECOMES SMALLER THAN -32768
:EXAMPLE:
   C:X=-32767-10
T:
T:
T:TYPE 'R' TO REPEAT ERROR MESSAGES.
   INMAX:1
A:
   INMAX:64
M:R
 JY:*ERRORS
   CH:
T:AN ILLEGAL STATEMENT IS DISPLAYED:
   MW:*
T:MW: IS NOT LEGAL IN THIS SYSTEM.
T:
T:YOU CAN OBTAIN IMMEDIATE OPERATION OF ANY INSTRUCTION BY
T:PRECEDING IT WITH '\'.   FOR EXAMPLE:
T:   \J:*LABEL  WILL JUMP IMMEDIATELY TO *LABEL.
T:   \EDIT:  WILL ENTER THE EDIT MODE AT THE PRESENT LOCATION.
T:   \  FOLLOWED BY RETURN WILL RESTART PILOT IN COMMAND MODE.
FOOT:
*END
CH:
CA:5,20
T:  TESTS ARE NOW COMPLETE
CA:7
T:Would you like to see tests of some advanced features? (Y/N) \
AS:
M:Y
LOADY:PLTST2
FOOT:Press RETURN for command mode \
CH:
T:PLTST2        TESTS OF ADVANCED PILOT FEATURES:
T:
T:      C       Concatenation of strings in the C-statement
T:      I       Indirect string reference and forced accept (A:=)
T:      S       Substrings produced by successful matches
T:      A       Ambiguous (wild) strings * and ?
T:      M       Match jump statement (MJ:)
T:      J       Jump according to item in match list (JM:)
T:      V       VNEW:\$ and VNEW:\#
T:      E       EXIT TO COMMAND MODE
T:
T:              Select test with a single letter

*SELECT
CA:12,6
AS:                   (Accepts a single character without RETURN)
   M:C,I,S,A,M,J,V,E
   JM:*CONCAT,*INDIRECT,*SUBSTR,*AMBIG,*MJUMP,*JMATCH,*VNEW,*EX
   J:*SELECT

*CONCAT
LOAD:TST-C

*INDIRECT
LOAD:TST-I

*SUBSTR
LOAD:TST-S

*AMBIG
LOAD:TST-WILD

*MJUMP
LOAD:TST-MJ

*JMATCH
LOAD:TST-JM

*VNEW
LOAD:TST-VN

*EX E:
```

```
CH:
T:TST-C        Concatenation of strings in the C-statement
T:
T:TYPE A NUMBER
A:#A
T:TYPE  "WORD"
A:$NAME
T:NOW TYPE ANY WORD
A:$WORD
C:$TEST=THE NUMBER IS #A, THE WORD IS $$NAME.
T:TEST: $TEST
FOOT:
LOAD:PLTST2
-------------------------------------------------------------
CH:
T:TST-I        Indirect string reference and forced accept (A:=)
T:
T:ENTER 'END'
A:$ONE
T:ENTER 'ONE'
A:$TWO
T:ENTER 'TWO'
A:$THREE
T:THIS SHOULD BE END:    $$$THREE
T:THIS SHOULD BE END:    $ONE
T:THIS SHOULD BE ONE:    $TWO
T:THIS SHOULD BE END:    $$TWO
T:THIS SHOULD BE TWO:    $THREE
T:THIS SHOULD BE ONE:    $$THREE
T:THIS SHOULD BE END:    $$$THREE
A:=$$THREE
M:ONE
TY:YES, INDIRECTION WORKS FOR A:=
TN:OOPS!  SOMETHING'S WRONG IF YOU SEE THIS
FOOT:
LOAD:PLTST2
-------------------------------------------------------------
CH:
T:TST-S        Substrings produced by successful matches
*START
T:
T:TYPE SOMETHING WITH "THIS" IN THE MIDDLE
A:
M:THIS
TN:NO MATCH
Y:$LEFT
Y:$MATCH
Y:$RIGHT
T:
T:AGAIN? (Y/N)
AS:
M:Y
JY:*START
LOAD:PLTST2
```

```
CH:
T:TST-WILD     Ambiguous (wild) strings * and ?
T:
T:LOOK FOR A<anything>BC
A:
M:A*BC
U:*ANS

T:LOOK FOR A*BC
A:
M:A\*BC
U:*ANS

T:LOOK FOR *
A:
M:\*
U:*ANS

T:LOOK FOR A<any-one>BC
A:
M:A?BC
U:*ANS

T:LOOK FOR A?BC
A:
M:A\?BC
U:*ANS

T:LOOK FOR ?
A:
M:\?
U:*ANS

T:LOOK FOR A\BC
A:
M:A\BC
U:*ANS

LOAD:PLTST2

*ANS
 Y:MATCH
 N:NO MATCH
 A:
 E:
```

```
CH:
T:TST-MJ        Match jump statement (MJ:)
*START
T:
T:TYPE "ONE", "TWO", "THREE", OR "FOUR"
A:
MJ:ONE
T:ONE
J:*OUT
MJ:TWO
T:TWO
J:*OUT
MJ:THREE
T:THREE
J:*OUT
M:FOUR
T:FOUR

*OUT T:AGAIN? (Y/N) \
AS:
M:Y
JY:*START
LOAD:PLTST2
--------------------------------------------------------------
CH:
T:TST-JM        Jump according to item in match list (JM:)
*START
T:
T:TYPE ONE, TWO, OR THREE
A:
M:ONE,TWO,THREE
JM:*ONE,*TWO,*THREE
TY:NO MATCH
J:*AGAIN

*ONE T:ONE
  J:*AGAIN
*TWO T:TWO
  J:*AGAIN
*THREE T:THREE

*AGAIN
T:AGAIN? (Y/N)
AS:
M:Y
JY:*START
LOAD:PLTST2
```

```
CH:
T:TST-VN       VNEW:\$ and VNEW:\#
T:
T:Operation:           PILOT Code:          Expected:
T:                     C:N=2
T:                     C:\$NAME=THIS
T:                     T:\#N, \$NAME        2, THIS
C:N=2
C:$NAME=THIS
T:#N, $NAME

T:                     VNEW:\$
T:                     T:\#N, \$NAME        2, \$NAME
VNEW:$
T:#N, $NAME

T:                     C:\$NAME=THIS
T:                     VNEW:\#
T:                     T:\#N, \$NAME        0, THIS
C:$NAME=THIS
VNEW:#
T:#N, $NAME

FOOT:
LOAD:PLTST2
```

```
R:   W.A.P.P. - WRITE A PILOT PROGRAM
R:   M. KAMP       UCSF      MARCH 1978
R:   J. STARKWEATHER, DISK VERSION, APRIL 1979
CH:
CA:2
T:T H I S   I S   T H E   W. A. P. P.   P R O G R A M . . . . .
CA:5
T:   W.
T:
T:   A.
T:
T:   P.
T:
T:   P.
PA:2
CA:2
CL:
T:W. A. P. P.   S T A N D S   F O R . . . . .
PA:1
CA:5,12
T:W R I T E
CA:7,12
T:A
CA:9,12
T:P I L O T
CA:11,12
T:P R O G R A M
CA:7,40
T:WRITTEN BY
CA:8,40
T:MARTY KAMP, 3/78
CA:9,40
T:& JOHN STARKWEATHER, 4/79
FOOT:
*INSTR
CH:
CA:6,5
TH:DO YOU WANT INSTRUCTIONS ?  ( Y OR N )....
AS:
M: N ,
JY:*GO
M: Y ,
JN:*INSTR
CH:
CA:2
T:   THIS PROGRAM WILL PRODUCE A PILOT PROGRAM CONTAINING
T:
T:MULTIPLE CHOICE QUESTIONS AND RESPONSES.  YOU WILL BE ASKED
T:
T:TO TYPE IN THE TEXT OF THE QUESTION ( 1-3 LINES ), 2 TO 5
T:
T:ALTERNATIVE ANSWER CHOICES,  AND ONE OR TWO LINES OF FEEDBACK
T:
T:FOR EACH OF THE CHOICES.    W.A.P.P. WILL DO ALL THE
T:
```

```
T:PROGRAMMING AND HOUSEKEEPING FOR YOU.
FOOT:
CH:
CA:2
T:   Before you tell W.A.P.P. to save a section of the program
T:
T:on disk, make sure you have a disk in unit B.  W.A.P.P. will
T:
T:not save what you have written until you have a chance to
T:
T:review your text.  You can make changes as you type in each
T:
T:line, but (in this version) once you have entered a line of
T:
T:text you must re-enter the whole question if you need to make
T:
T:changes.
FOOT:
CH:
CA:2
T:   After you type in the text of a question along with its
T:
T:answers and feedback, you can tell W.A.P.P to save it on disk.
T:
T:W.A.P.P. will generate the PILOT code and position everything
T:
T:appropriately on the screen.  Each question is saved as a
T:
T:separate file which will LOAD the next question.  You can run
T:
T:the program produced by W.A.P.P. by typing "LOAD WAP1" or
T:
T:"GET WAP1" followed by "RUN".  The program may be further
T:
T:edited and then saved like any other PILOT program.
FOOT:
*GO
R:Initialize frame count
C:I=1
CH:
CA:2
T:   IT IS POSSIBLE TO SET UP YOUR PROGRAM SO IT WILL COLLECT
T:
T:ALL RESPONSES AND WRITE THEM INTO A DISK FILE (DATA).
T:
T:IF YOU TYPE  'Y'  W.A.P.P. WILL BUILD THIS
T:
TH:CAPABILITY INTO YOUR PROGRAM, OTHERWISE HIT "RETURN"...
AS:
M: Y , YES ,
R:   IF K=1 PROGRAM WILL CONTAIN OPENF: AND WRITE: COMMANDS
CY:K=1
R:  L IS USED LATER TO OPENF: ONLY ONCE FOR DATA COLLECTION
C:L=1
CAY:10
```

```
TY:   OK, DATA WILL BE COLLECTED AND SAVED ON FILE "DATA".
FOOTY:
*BEGIN
CH:
R:   RESET TEXT VARIABLES
VNEW:$
R:   RESET DISK SAVE FLAG
C:U=0
T:YOU MAY ENTER UP TO THREE LINES OF TEXT.
CA:4
T:ENTER YOUR FIRST LINE OF TEXT.
CA:6
A:$T1
CA:4
CL:
C:M=2
T:ENTER YOUR SECOND LINE OF TEXT.
CA:7
A:$T2
C:M=3
C:F=0
C:G=0
R:   M WILL BE LINE WHERE CHOICES START
U:*ALPHA
CN:M=2
JN:*CHOICES
C:F=1
CA:4
CL:
T:ENTER YOUR THIRD LINE OF TEXT.
CA:8
A:$T3
C:M=4
U:*ALPHA
CN:M=3
CY:G=1
J:*CHOICES
*ALPHA
R:   MATCH ANY LETTER
M:E,T,O,I,N,S,H,A,U,R,Y,B,C,D,F,G,J,K,L,M,P,Q,V,W,X,Z,1,2,3,4,5,
MN:6,7,8,9,0,
E:
*CHOICES
CH:
C:Z=M-1
T(Z):$T1
C:Z=M-2
T(Z):$T2
C:Z=M-3
T(Z):$T3
CA:6
T:HERE IS HOW YOUR TEXT
T:WILL APPEAR ON THE SCREEN.....
*ENTERC
CA:9
```

```
CE:
C:P=0
T:HOW MANY CHOICES IN THIS QUESTION?
T:  ENTER A NUMBER, 2 - 5.
AS:#N
M:2,3,4,5,
JN:*ENTERC
M: 2 ,
CY:P=P+1
M: 3 ,
CY:P=P+1
M: 4 ,
CY:P=P+1
M: 5 ,
CY:P=P+1
J(P):*EC2
J:*ENTERC
*EC2
C:P=P-1
J(P):*ENTERC
R:   N = NUMBER OF CHOICES REQUESTED
R:   M = ROW WHERE CHOICE 1 WILL BE DISPLAYED
R:   O = CURRENT CHOICE NUMBER
R:   P = CURRENT ROW NUMBER
C:O=1
C:P=M
CA:M
CE:
*CLOOP
CA:10
CE:
T:ENTER CHOICE NUMBER #O.
CA:P
T:   (#O)
CA:P,8
R:   *CINPUT ACCEPTS AND STORES TEXT OF CHOICES
U:*CINPUT
R:   CHECK TO SEE IF N CHOICES HAVE BEEN ENTERED.
C:O=O+1
C:Q=O-N
J(Q):*GOOD
R:   MORE CHOICES NEEDED
C:P=P+1
J:*CLOOP
*CINPUT
R:   ACCEPTS TEXT FOR CHOICE #O  ( 1-5 )
C:Q=O-4
A(Q):$C5
E(Q):
C:Q=O-3
A(Q):$C4
E(Q):
C:Q=O-2
A(Q):$C3
E(Q):
```

```
C:Q=O-1
A(Q):$C2
E(Q):
A:$C1
E:
*GOOD
C:A=0
C:B=0
C:C=0
C:D=0
C:E=0
C:X=0
R:   SET J EQUAL TO NUMBER OF CHOICES
C:J=N
CA:9
CE:
CA:10
T:WHICH CHOICE IS CORRECT?  ( 1 TO #N )
CA:10,40
AS:
R:    Y IS THE NUMBER OF THE CORRECT CHOICE
M: 1 ,
CY:A=1
CY:Y=1
U(J):*CR
M: 2 ,
CY:B=1
CY:Y=2
U(J):*CR
M: 3 ,
CY:C=1
CY:Y=3
U(J):*CR
M: 4 ,
CY:D=1
CY:Y=4
U(J):*CR
M: 5 ,
CY:E=1
CY:Y=5
U(J):*CR
J(X):*CH1
C:X=2
*CH1
C:X=X-1
J(X):*GOOD
J:*FEEDBACK
*CR
CY:X=X+1
C:J=J-1
E:
*FEEDBACK
R:    COLLECT FEEDBACK TEXT
R:    Q IS CHOICE COUNTER
C:Q=1
```

```
R:     R = FIRST ROW AFTER CHOICES
C:R=M+N
R:     P = LAST ROW OF TEXT
C:P=M-1
R:     S = CHOICE LINE TO BE *** MARKED
C:S=0
*FLOOP
R:     REMOVE STARS
CA:S
T(S):    (
CA:R
CE:
CA:13
T:ENTER FIRST LINE OF FEEDBACK FOR CHOICE #Q
C:S=P+Q
R:     MARK CHOICE #Q, WHICH IS ON ROW #S
CA:S
T:***
R:     COLLECT FIRST LINE ON ROW #T
C:T=R+1
CA:T
U:*FINPUT1
CA:13
CL:
T:ENTER SECOND LINE OF FEEDBACK FOR CHOICE #Q
C:T=R+2
CA:T
U:*FINPUT2
R:     SEE IF WE HAVE DONE N CHOICES
C:Q=Q+1
C:X=Q-N
J(X):*LABEL5
J:*FLOOP
*FINPUT1
R:     COLLECTS FIRST LINE OF FEEDBACK FOR CHOICE #Q
C:W=Q-4
A(W):$FB5A
E(W):
C:W=Q-3
A(W):$FB4A
E(W):
C:W=Q-2
A(W):$FB3A
E(W):
C:W=Q-1
A(W):$FB2A
E(W):
A:$FB1A
E:
*FINPUT2
R:     COLLECTS SECOND LINE OF FEEDBACK FOR CHOICE #Q
C:W=Q-4
A(W):$FB5B
E(W):
C:W=Q-3
```

```
A(W):$FB4B
E(W):
C:W=Q-2
A(W):$FB3B
E(W):
C:W=Q-1
A(W):$FB2B
E(W):
A:$FB1B
E:
*LABEL5
CH:
T:PRESS Q TO DISPLAY THE QUESTION AND THE CHOICES.
CA:3
T:PRESS F TO DISPLAY THE FEEDBACK FOR THE CHOICES.
CA:5
T:PRESS D TO WRITE THIS SECTION OF PROGRAM ON THE DISK.
CA:7
T:PRESS A TO WRITE ANOTHER FRAME.
CA:9
T:PRESS E TO END THIS PROGRAM
CA:11
AS:
M: Q ,
JN:*LX1
U:*DISPLAYQ
J:*LABEL5
*LX1 M: F ,
JN:*LX2
U:*DISPLAYF
J:*LABEL5
*LX2 M: D ,
JN:*LX3
U:*DISK
J:*LABEL5
*LX3 M: E ,
JY:*END
M: A ,
JN:*LABEL5
R:IF DISK STORAGE HAS OCCURRED, THEN NEXT QUESTION
J(U):*BEGIN
CH:
CA:4
T:***   WARNING - THIS COMMAND ERASES YOUR STORED TEXT FROM
T:***   THE COMPUTER MEMORY...  MAKE SURE YOU HAVE STORED IT
T:***   ON THE DISK IF YOU WANT TO KEEP IT.
T:***
T:***   IF YOU WANT TO SAVE YOUR PROGRAM ON THE DISK, TYPE "D",
TH:***   OTHERWISE "RETURN" TO START ON A NEW QUESTION...
AS:
M: D
UY:*DISK
J:*BEGIN
*DISPLAYQ
CH:
```

```
CA:3
T:$T1
R:   DON'T DISPLAY $T2 OR $T3 IF NO TEXT THERE
T(F):$T2
T(G):$T3
C:Q=N
T(Q):    (1) $C1
C:Q=N-1
T(Q):    (2) $C2
C:Q=N-2
T(Q):    (3) $C3
C:Q=N-3
T(Q):    (4) $C4
C:Q=N-4
T(Q):    (5) $C5
FOOT:
E:
*DISPLAYF
CH:
CA:3
C:Q=N
T(Q):1. $FB1A
T(Q):   $FB1B
C:Q=N-1
T(Q):2. $FB2A
T(Q):   $FB2B
C:Q=N-2
T(Q):3. $FB3A
T(Q):   $FB3B
C:Q=N-3
T(Q):4. $FB4A
T(Q):   $FB4B
C:Q=N-4
T(Q):5. $FB5A
T(Q):   $FB5B
FOOT:
E:
*DISK
R:  OPEN FILE TO WRITE PILOT PROGRAM
CREATEF:WAP#I.PLT
OPENF:WAP#I.PLT,H
RW:H,R: PROGRAM PRODUCED BY W.A.P.P
R: PUT OPENF: ON DISK IF K=1 (DATA DESIRED)
U(L):*OPNF
J:*OPNF9
*OPNF
RW(K):H,R:  OPENF ONLY ON FIRST FRAME
RW(K):H,CREATEF:DATA
RW(K):H,OPENF:DATA,V
R: SET L=0 SO NO MORE OPENF.
C:L=0
E:
*OPNF9
RW(K):H,RW:V,QUESTION #I  ( ANS. = #Y )
RW:H,*START
```

```
RW:H,CH:
RW:H,INMAX:1
R:   WRITE (M-1) LINES OF TEXT ON DISK
C:Q=M-1
RW(Q):H,T:$T1
C:Q=M-2
RW(Q):H,T:$T2
C:Q=M-3
RW(Q):H,T:$T3
R:   WRITE N CHOICES ON DISK
RW:H,T:    (1) $C1
C:Q=N-1
RW(Q):H,T:    (2) $C2
C:Q=N-2
RW(Q):H,T:    (3) $C3
C:Q=N-3
RW(Q):H,T:    (4) $C4
C:Q=N-4
RW(Q):H,T:    (5) $C5
RW:H,*ANS
R:   SKIP A LINE, PUT A-STATEMENT ON SCREEN
C:Q=M+N
C:Q=Q+1
R:   DON'T ACCEPT ANSWER ANY LOWER THAN ROW 9
C:W=Q-9
C(W):Q=9
RW:H,CA:#Q
RW:H,CL:
RW:H,A:
RW(K):H,WRITE:V
RW:H,CE:
C:Q=Q+2
R:   DON'T BEGIN FEEDBACK LOWER THAN ROW 10
C:W=Q-10
C(W):Q=10
RW:H,CA:#Q
RW:H,C:X=0
R:   CHECK FOR LEGAL ANSWER
RW:H,C:J=#N
RW:H,M(J): 1 ,
RW:H,U(J):*TD
RW:H,M(J): 2 ,
RW:H,U(J):*TD
RW:H,M(J): 3 ,
RW:H,U(J):*TD
RW:H,M(J): 4 ,
RW:H,U(J):*TD
RW:H,M(J): 5 ,
RW:H,U(J):*TD
RW:H,J(X):*TD1
RW:H,C:X=2
RW:H,*TD1
RW:H,C:X=X-1
RW:H,T(X):PLEASE TYPE A NUMBER, 1 TO #N.
RW:H,J(X):*ANS
```

```
RW:H,J:*OK
RW:H,*TD
RW:H,CY:X=X+1
RW:H,C:J=J-1
RW:H,E:
RW:H,*OK
R:   ANSWER IS LEGAL
R:   ABCDE REVEAL THE CORRECT ANSWER
R:   Y IS NUMBER OF CORRECT ANSWER
R:   N IS NUMBER OF CHOICES
C:W=M
RW:H,M: 1 ,
RW:H,JN:*L2
RW:H,CA:#W
RW:H,T:***
RW:H,PA:1
RW:H,CA:#Q
RW:H,T:$FB1A
RW:H,T:$FB1B
RW(A):H,J:*DONE
J(A):*LABEL7
RW:H,J:*ANS
*LABEL7
RW:H,*L2
C:W=M+1
RW:H,M: 2 ,
RW:H,JN:*L3
RW:H,CA:#W
RW:H,T:***
RW:H,PA:1
RW:H,CA:#Q
RW:H,T:$FB2A
RW:H,T:$FB2B
RW(B):H,J:*DONE
J(B):*LABEL8
RW:H,J:*ANS
*LABEL8
RW:H,*L3
R:   QUIT IF ONLY 2 ANSWERS ( N=2 )
C:X=3
C:X=X-N
J(X):*NOMORE
C:W=M+2
RW:H,M: 3 ,
RW:H,JN:*L4
RW:H,CA:#W
RW:H,T:***
RW:H,PA:1
RW:H,CA:#Q
RW:H,T:$FB3A
RW:H,T:$FB3B
RW(C):H,J:*DONE
J(C):*LABEL9
RW:H,J:*ANS
*LABEL9
```

```
RW:H,*L4
R:   QUIT IF ONLY THREE ANSWERS ( N=3 )
C:X=4
C:X=X-N
J(X):*NOMORE
C:W=M+3
RW:H,M: 4 ,
RW:H,JN:*L5
RW:H,CA:#W
RW:H,T:***
RW:H,PA:1
RW:H,CA:#Q
RW:H,T:$FB4A
RW:H,T:$FB4B
RW(D):H,J:*DONE
J(D):*LABEL10
RW:H,J:*ANS
*LABEL10
RW:H,*L5
R:   QUIT IF ONLY 4 ANSWERS  ( N=4 )
C:X=5
C:X=X-N
J(X):*NOMORE
C:W=M+4
RW:H,M: 5 ,
RW:H,JN:*DONE
RW:H,CA:#W
RW:H,T:***
RW:H,PA:1
RW:H,CA:#Q
RW:H,T:$FB5A
RW:H,T:$FB5B
RW(E):H,J:*DONE
J(E):*LABEL11
RW:H,J:*ANS
*LABEL11
*NOMORE
RW:H,*DONE
RW:H,CA:14
RW:H,CL:
RW:H,T:ANSWER  #Y  IS CORRECT
RW:H,*AGAIN
RW:H,CA:16
RW:H,CL:
RW:H,TH:DO YOU WANT TO REPEAT THIS QUESTION ?  ( Y OR N )...
RW:H,A:
RW:H,M: Y ,
RW:H,JY:*START
RW:H,M: N ,
RW:H,JN:*AGAIN
C:I=I+1
RW:H,LOAD:WAP#I.PLT
EOF:H
CLOSEF:H
C:U=1
```

```
   E:  END OF *DISK ROUTINE
*END
   R:END OF WAPP PROGRAM
J(U):*END2
T:*** IF YOU WISH TO SAVE THE LAST PROGRAM SECTION, TYPE "D"
T:*** OTHERWISE "RETURN" TO END WAPP OPERATION.
INMAX:1
A:
M: D
UY:*DISK
*END2
CREATEF:WAP#I.PLT
OPENF:WAP#I.PLT,H
RW(K):H,EOF:V
RW(K):H,CLOSEF:V
EOF:H
CLOSEF:H
E:
     R: HODGE PODGE HOMONYMES  S.WILLIAMS & K.ANDERSON
     R: MOD 5/75 & 10/77, J. STARKWEATHER

   CH:
*START
C:A=0
C:B=0
C:C=0
C:D=0
C:E=0
C:F=0
C:G=0
C:H=0
C:I=0
C:J=0
C:L=1
C:X=0
U(P):*DISPLAY
J(P):*PASS
C:P=1
   CA:2,48
T:
   CA:5
T:THIS PROGRAM WILL SHOW YOU ROWS OF LETTERS.
T:
T:PRESS 'RETURN' TO SEE THEM.
  INMAX:1
  A:
U:*DISPLAY
PA:5
   CH:
   CA:3
T:YOU WILL TRY TO FIND WORDS THAT ARE IN PAIRS - HOMONYMS.
T:
T:THE WORDS SOUND THE SAME, BUT THEY HAVE DIFFERENT MEANING.
T:
T:ONE OF EACH PAIR IS WRITTEN ACROSS, THE OTHER UP AND DOWN.
T:
T:LOOK FIRST FOR THE WORDS ACROSS.
T:
T:I SAW THE WORDS 'HOUR' ACROSS, AND 'OUR' READING DOWN.
T:
T:       PRESS "RETURN" AND I WILL SHOW YOU.
A:
C:Y=8
U:*DISPLAY
U:*BLINK
*PASS
   CA:11
T:TYPE THE WORDS YOU SEE.
T:TRY AGAIN AND AGAIN - I AM KEEPING SCORE TO SEE HOW MANY
T:YOU CAN FIND IN 18 TRIES.    THERE ARE 18 WORDS.
T:TYPE ONLY ONE WORD AT AT TIME, AND THEN PRESS 'RETURN'.
T:YOU CAN ERASE BAD TYPING WITH THE 'DEL' KEY.
T:   NOW PRESS 'RETURN' TO START.
```

```
   CA:16,33
A:
   CA:11
   CE:

*ANS
   CA:12
   CL:
INMAX:64
A:
C:A=A+1
U:*TEST
C:K=A-17
J(K):*DONE
J:*ANS

*DONE
   CA:11
   CE:
T:YOU HAVE HAD 18 TRIES, PRESS "RETURN" AND
T:I WILL SHOW YOU YOUR SCORE.
A:
   CA:11
   CE:
T:YOUR SCORE IS #X OUT OF A POSSIBLE 18.
T:PRESS "\" TO QUIT, OR "RETURN" TO TRY AGAIN.
   CA:12,55
INMAX:1
A:
   CH:
J:*START

*DISPLAY
   CH:
   CA:1,36
T:X Q R A U N T T N R Z U C Q
   CA:2,36
T:V V A N E V W Z H O U R P O
   CA:3,36
T:E U V A X Y Z I I N Q R R U
   CA:4,36
T:I P Q N R S T N U V B Y I R
   CA:5,36
T:N W X T Y Z A N M E E T N E
   CA:6,36
T:Q C D Q F G R H I J B K C M
   CA:7,36
T:O W R A P L A M S S U N I E
   CA:8,36
T:P Q R S T U P V Q E Y W P A
   CA:9,36
T:P R I N C I P L E E X Y A T
   CA:10,36
T:S C E N E Z A B N N C D L T
   R: AUNT/ANT - VANE/VEIN - HOUR/OUR - PRINCIPLE/PRINCIPAL
   R: SEEN/SCENE
   R: WRAP/RAP - BY/BUY - LITE/LIGHT - IN/INN
E:

*BLINK
   CA:2,50
T:. . . .
   CA:2,62
T:.
   CA:3,62
T:.
   CA:4,62
T:.
U:*NULL
   CA:2,50
T:H O U R
   CA:2,62
T:O
   CA:3,62
T:U
   CA:4,62
T:R
U:*NULL
C:Y=Y-1
J(Y):*BLINK
E:

*TEST
*M1 M:AUNT
JN:*M2
   CA:1,42
T:. . . .
J(B):*M1A
C:B=L
C:L=L+1
*M1A CA:B
T:AUNT
J:*EOT

*M2 M:VANE
JN:*M3
   CA:2,38
T:. . . .
J(C):*M2A
C:C=L
C:L=L+1
*M2A CA:C
T:VANE
J:*EOT

*M3 M:HOUR
JN:*M4
   CA:2,52
TY:. . . .
J(D):*M3A
```

```
C:D=L
C:L=L+1
*M3A CA:D
T:HOUR
J:*EOT

*M4 M: IN ,
JN:*M5
   CA:3,52
T:. .
J(E):*M4A
C:E=L
C:L=L+1
*M4A CA:E
T:IN
J:*EOT

*M5 M:BY
JN:*M6
   CA:4,56
T:. .
J(F):*M5A
C:F=L
C:L=L+1
*M5A CA:F
T:BY
J:*EOT

*M6 M:MEET
JN:*M7
   CA:5,52
T:. . . .
J(G):*M6A
C:G=L
C:L=L+1
*M6A CA:G
T:MEET
J:*EOT

*M7 M:WRAP
JN:*M8
   CA:7,37
T:. . . .
J(H):*M7A
C:H=L
C:L=L+1
*M7A CA:H
T:WRAP
J:*EOT

*M8 M:PRINCIPLE
JN:*M9
   CA:9,36
T:. . . . . . . . . .
J(I):*M8A
```

```
C:I=L
C:L=L+1
*M8A CA:I
T:PRINCIPLE
J:*EOT

*M9 M:SCENE
JN:*M10
   CA:10,36
T:. . . . .
J(J):*M9A
C:J=L
C:L=L+1
*M9A CA:J
T:SCENE
J:*EOT

*M10

*MD1 M:ANT
JN:*MD2
   CA:3,42
T:.
   CA:4,42
T:.
   CA:5,42
T:.
   CA:3,20
J(B):*MD1A
C:B=L
C:L=L+1
*MD1A CA:B,20
T:ANT
J:*EOT

*MD2 M:VEIN
JN:*MD3
   CA:2,36
T:.
   CA:3,36
T:.
   CA:4,36
T:.
   CA:5,36
T:.
   CA:2,20
J(C):*MD2A
C:C=L
C:L=L+1
*MD2A CA:C,20
T:VEIN
J:*EOT

*MD3 M: OUR
JN:*MD4
```

```
   CA:2,62
T:.
   CA:3,62
T:.
   CA:4,62
T:.
   CA:2,20
J(D):*MD3A
C:D=L
C:L=L+1
*MD3A CA:D,20
T:OUR
J:*EOT

*MD4 M:INN
JN:*MD5
   CA:3,50
T:.
   CA:4,50
T:.
   CA:5,50
T:.
   CA:6,50
J(E):*MD4A
C:E=L
C:L=L+1
*MD4A CA:E,20
T:INN
J:*EOT

*MD5 M:BUY
JN:*MD6
   CA:6,56
T:.
   CA:7,56
T:.
   CA:8,56
T:.
   CA:6,20
J(F):*MD5A
C:F=L
C:L=L+1
*MD5A CA:F,20
T:BUY
J:*EOT

*MD6 M:MEAT
JN:*MD7
   CA:6,62
T:.
   CA:7,62
T:.
   CA:8,62
T:.
   CA:9,62
```

```
T:.
   CA:6,20
J(G):*MD6A
C:G=L
C:L=L+1
*MD6A CA:G,20
T:MEAT
J:*EOT

*MD7 M:RAP
JN:*MD8
   CA:6,48
T:.
   CA:7,48
T:.
   CA:8,48
T:.
   CA:6,20
J(H):*MD7A
C:H=L
C:L=L+1
*MD7A CA:H,20
T:RAP
J:*EOT

*MD8 M:PRINCIPAL
JN:*MD9
   CA:2,60
T:.
   CA:3,60
T:.
   CA:4,60
T:.
   CA:5,60
T:.
   CA:6,60
T:.
   CA:7,60
T:.
   CA:8,60
T:.
   CA:9,60
T:.
   CA:10,60
T:.
   CA:2,20
J(I):*MD8A
C:I=L
C:L=L+1
*MD8A CA:I,20
T:PRINCIPAL
J:*EOT

*MD9 M:SEEN
JN:*MD10
```

```
   CA:7,54
T:.
   CA:8,54
T:.
   CA:9,54
T:.
   CA:10,54
T:.
   CA:7;20
J(J):*MD9A
C:J=L
C:L=L+1
*MD9A CA:J,20
T:SEEN
J:*EOT

*MD10    R: NO MATCH FOUND */
E:

*EOT     R: A MATCH WAS FOUND */
C:X=X+1
E:

*NULL E:
```

SECTION 7

CONFIGURING AN UNKNOWN TERMINAL

In many cases you will find the terminal you are using listed for a simple choice. If it is not listed, you will be asked some questions about it and this may be sufficient to get PILOT up and running. This is an example of one such interaction. In this case the responses are those for a Zenith Z-19 terminal and this method would not be required. For further customizing of special cases, the file NVPILOT.PRN provides additional information.

```
A>NVPILOT

PILOT Version 5.0 configuring
Copyright (C) 1981,1982, J. Starkweather
@ ANSI Mode Terminal
A Apple Computer, 40 column display
V Apple Computer + VIDEX 80 column board
B Beehive 150 or Cromemco 3100
C Flashwriter II, memory at F000H
D Hazeltine 1400 series
E Hazeltine 1500 series
F Heath H19/H89 or Zenith Z19/Z89
G Hewlett-Packard 2621
H IBM Personal Computer + Baby Blue Card
I Infoton I-100

Type a single letter to select terminal.
<Carriage Return> for more terminals

J Imsai VIO, memory at F000H
K Lear-Seigler ADM-3A
L Lear-Seigler ADM-31
M Microterm ACT-IV
N Osborne I
O Perkin-Elmer 550 (Bantam)
P Processor Technology Sol or VDM
Q Soroc IQ-120/140
R SuperBrain
S Televideo
T TRS-80, Mod. II (P.& T. CP/M)
U None of the above

  Type a single letter to select terminal.
  <Carriage Return> for more terminals U

  Enter 2 digits for number of lines
  in the display. 24

  Enter 2 digits for number of characters
  per line. ('U' will restart entries.) 80
```

Section 8: Configuring example.

```
Enter M for memory-mapped display, T for
a serial-connected terminal.  (M or T) T

Most terminals position the cursor from
a sequence of characters as follows:
   Lead characters, differing for
     different terminals.
   The line number, sometimes offset
   Sometimes separator characters
   The column number, sometimes offset
   Sometimes ending characters
On some terminals column is before line.
Does your terminal follow
this general pattern? (Y/N) Y

Enter no. of lead characters
for cursor positioning. 2

Enter the first lead character
in hex, e.g. '1B'. 1B
Enter the next lead character. 59

Enter the no. of line/col separator
characters. 0

Enter the no. of ending characters. 0

Enter offset to be added to line value.
   Enter 2 hex characters, e.g. 20 20

Enter offset added to column value. 20

Is column entered before line? (Y/N) N

Following controls will speed editing.
If control is not available, enter zero.

Enter no. of characters to clear screen
and home the cursor to upper left. 2
Enter two hex characters for each. 1B45

Enter no. of characters to insert line
above cursor position. 2
Enter two hex characters for each. 1B4C

Enter the no. of characters to delete
the cursor line. 2
Enter two hex characters for each. 1B4D

Enter the no. of characters to
turn on reverse video. 2
Enter two hex characters for each. 1B70
Enter the no. of characters to
turn off reverse video. 2
Enter two hex characters for each. 1B71

Enter the no. of characters to
reset the terminal. 2
Enter two hex characters for each. 1B7A

Enter the no. of characters to
turn the cursor on. 3
Enter two hex characters for each. 1B7935

Enter the no. of characters to
turn the cursor off. 3
Enter two hex characters for each. 1B7835

Enter 'A' for full PILOT for AUTHOR. A
Enter 'P' to limit DIR to *.PLT files. P

Configuring of PILOT.COM is complete.
...Now saving it.

PILOT.COM saved on the default drive.

A>
```

DISCLAIMER

All Ellis Computing computer programs are distributed on an "AS IS" basis without warranty.

ELLIS COMPUTING makes no warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. In no event will ELLIS COMPUTING be liable for consequential damages even if ELLIS COMPUTING has been advised of the possibility of such damages.

* available in immediate mode

INDEX TO PILOT STATEMENTS AND COMMANDS

Starkweather, J.A. Computest: a computer language for individualized testing, instruction, and interviewing. Psychol. Rep. **17,** 227, 1965.

Office of Information Systems, PILOT 1.6 Guide. Report UCS 03.01.01, University of California, San Francisco, 1973.

Starkweather, J.A. A common language for a variety of conversational programming needs. Readings in Computer-assisted Instruction, edited by H.A. Wilson and R.C. Atkinson, p. 269. Academic Press, New York, 1969.

Starkweatner, J.A.,Kamp, M. and Monto, A. Psychiatric interview simulation by computer. Meth. Inform. Med. **6,** 15, 1967.

Kamp, M. Evaluating the operation of interactive free-response computer programs. J. Biomed. Systems, **2,** 33, 1971.

Kamp, M. and Starkweather, J.A. A return to a dedicated machine for computer-assisted instruction. Comput. Biol. Med. **3,** 293, 1973.

Ellis, C. NEVADA COBOL, Ellis Computing, 1980.

Ellis, C. NEVADA FORTRAN, Ellis Computing, 1982.

Ellis, C. and Starkweather, J.A. NEVADA EDIT, Ellis Computing, 1982.