TEXAS INSTRUMENTS
HOME COMPUTER

TI PILOT

Requires: Peripheral Expansion System, Memory Expansion Card, UCSD p-System* (P-Code Card and Editor/Filer/Utilities Diskette), and Disk Memory System (Disk Drive Controller Card and at least one disk drive). Optional Solid State Speech™ Synthesizer required to use speech capabilities.

A programming language designed especially for the development of Computer Assisted Instruction programs with the TI Home Computer.

(inside front cover)

Quick Guide to II PILOT

(proposed cover copy)

TEXAS INSTRUMENTS
HOME COMPUTER

TI PILOT

Requires: Peripheral Expansion System, Memory Expansion Card, UCSD p-System* (P-Code Card and Editor/Filer/Utilities Diskette), and Disk Memory System (Disk Drive Controller Card and at least one disk drive). Optional Solid State Speech™ Synthesizer required to use speech capabilities.

A programming language designed especially for the development of Computer Assisted Instruction programs with the TI Home Computer.

UCSD p-System is a trademark of the
Regents of the University of California.

(Title page)

TI PILOT

TI PILOT
Table of Contents

SECTION 1: GENERAL INFORMATION

The use of computer-based, interactive instructional materials has grown widely in recent years. As the use of computers has spread throughout the educational spectrum, the need for a computer programming language that can be readily learned by instructors has grown accordingly. PILOT is one of the languages that have been developed to fill that need. The name PILOT is an acronymn for Programmed Inquiry, Learning, Or Teaching, and the language originated in the early 1970s to simplify the creation and delivery of CAI programs.

TI PILOT is a language designed specifically for computer-assisted instruction (CAI). With TI PILOT, you can run existing programs written in the TI PILOT language, or use the UCSD p-System* for the TI Home Computer to:

- Develop individualized and interactive instructional programs which teach a student a specific subject.
- Develop drill-and-practice programs to reinforce concepts taught in the classroom.
- Develop testing programs which automatically score students and chart the progress of a class.
- Store a variety of information including a student's performance, progress in a course, and attendance history.

With TI PILOT, you can develop effective educational programs even if you have had little or no programming experience.

The term CAI covers a variety of educational programs, including testing, record keeping, demonstration of concepts, individualized drill and practice for reinforcement and remediation, and simulations that create a laboratory-like environment. These programs can provide teachers with effective classroom tools.

TI PILOT is an expanded version of PILOT with enhancements that let you take advantage of unique features of the TI Home Computer such as speech, sound, and graphics. The language is designed to be compatible with the UCSD p-System (sold separately) developed for the TI Home Computer. (See section 1.1).

The diskette enclosed in the TI PILOT package contains the TI PILOT Interpreter, which allows the programs you develop to be run ("executed") on the TI Home Computer. Also included on the diskette is a series of sample programs that you can study and practice with while you are learning.

* UCSD p-System is a trademark of the Regents of the University of California.

1.1 Equipment Requirements

In addition to the TI Home Computer and the TI Color Monitor (or the TI Video Modulator and a television set), the following equipment and software are required for developing and running TI PILOT programs.

- TI Peripheral Expansion System.
- TI Memory Expansion Card.
- TI P-Code Card.
- TI Disk Memory System (the TI Disk Drive Controller Card and at least one TI Disk Memory Drive).
- UCSD p-System Editor/Filer/Utilities Diskette (required for program development).
- TI Solid State Speech™ Synthesizer, if you want to include speech in your program.
- TI PILOT Diskette.

You will find it convenient to have at least one additional diskette to save the programs and data you develop. A printer can be attached to your system to increase its capabilities.

1.2 Using this Manual

This manual assumes that you are not familiar with the PILOT programming language. You can learn the language and the fundamental steps for creating a TI PILOT program by running the demonstration programs included on the TI PILOT diskette, working through the tutorial section (section 3), and then reading the remainder of the manual and studying the examples included there.

As you study the instructions in sections 3 and 4, enter and run the program examples which are included in those sections and then experiment with the instructions by making changes to the programs.

An attempt has been made to acquaint you with p-System procedures in this manual. For more details concerning these procedures, refer to the Editor/Filer/Utilities manuals.

The following sections are included in this manual.

Section	Contents
1. General Information	Contains general information about TI PILOT, including an overview of the process of developing a TI PILOT program using the UCSD p-System and a description of the special keys used with the UCSD p-System and TI PILOT.
2. Getting Started	Describes the procedures for setting up the system and for entering and running a TI PILOT program. Step-by-step instructions are given for running the sample programs on the TI PILOT diskette.
3. TI PILOT Tutorial	Directs the beginning programmer through a series of sample activities in order to become acquainted with the fundamental instructions of TI PILOT in both the Immediate Mode and the development of a practice program.
4. The Format of a TI PILOT Instruction	Contains a description of the format of a TI PILOT instruction and an overview of the types of data used in a TI PILOT program.
5. TI PILOT Instructions	Describes each of the language's instructions. Numerous examples are included to illustrate the operation of the instructions.
6. Student Commands	Describes the two student commands which can be used when a program is running. These commands allow a student to control the operation of a program and are useful also for testing a program you have written.
7. Data Types	Describes in detail the kinds of data used by a TI PILOT program with examples of each of the data types.
8. Functions	Describes the numeric and string functions available with TI PILOT, including examples of the functions.
9. Errors	Describes the error messages which can result from running a program.
10. Programming Recommendations	Contains recommendations for creating effective TI PILOT programs.

The remainder of the manual contains suggested procedures should you encounter problems, a number of appendices which can be useful when you are developing programs, an index listing references to key words and topics discussed in the manual, and warranty information.

1.3 Developing a TI PILOT Program

Briefly, this is the procedure for developing a TI PILOT program.

1. Define the program--Define the purpose of the program, and determine the procedure it will follow for accomplishing its purpose.

2. Write the program--First, write on paper the instructions which make up the program. Then, after you have reviewed the program, use the p-System Editor to enter the program. The Editor stores the program in a TEXT file on a diskette.

3. Run the program--First, execute the TI PILOT Interpreter. The interpreter asks you for the name of the course (the program). Enter the name of the course and TI PILOT automatically loads the program into memory, translates the instructions, and performs them.

4. Test the Program--When the program runs, test it to make sure it is working correctly.

5. Modify the Program--If you want to correct mistakes or make other changes to the program after running it, use the Editor to modify the TEXT file. Then run the program again to check your changes.

6. Document the Program--To help others understand the program (and to help you understand it after you have been away from it for a while), describe the purpose of the program and how it works.

1.4 Special Keys

Note that the < and > symbols indicate function keys to be pressed and not information to be typed. The name <return> is used when the TI PILOT messages (prompts) on the screen refer to <return> or <cr> (carriage return). Press the ENTER key for <return> or <cr>.

To type lower-case letters, press the key with the letter on it. Pressing any key for more than approximately half a second causes that key to be repeated until the key is released.

To type all upper-case letters on the TI-99/4, use the alpha lock toggle to change to upper-case. On the TI-99/4A you may use the alpha lock toggle or press the ALPHA LOCK key.

To type a single upper-case letter on the TI-99/4 when the computer is in lower-case mode, simultaneously press the small space key on the left side of the keyboard or the SPACE BAR and the key. On the TI-99/4A, press SHIFT and the key.

If you have a TI-99/4A console, it is suggested that you use one of the blank overlays supplied with your computer to make a special overlay for TI PILOT. This can serve as a reminder for operations done with both a number key and the FCTN key (or the CTRL key) such as <screen left> and <screen right>.

Here is a reference chart for special key functions used with the UCSD p-System. (This is also in the P-Code Card and Editor/Filer/Utilities manuals.)

Name	II-99/4	II-99/4A	Action
<del>	SHIFT F	FCTN 1	Deletes a character.
<ins>	SHIFT G	FCTN 2	Inserts a character.
<flush>	SPACE 3	FCTN 3	Stops writing output to the screen.
<break>	SPACE 4	FCTN 4	Stops the program and initializes the System.
<stop>	SPACE 5	FCTN 5	Suspends the program until this key is pressed again.
<alpha lock>	SPACE 6	FCTN 6 or ALPHA LOCK	Acts as a toggle to convert upper-case letters to lower-case and back again.
<screen left>	SPACE 7	FCTN 7	Moves the text displayed on the screen to the left 20 columns at a time.
<screen right>	SPACE 8	FCTN 8	Moves the text displayed on the screen to the right 20 columns at a time.
<line del>	SHIFT Z	FCTN 9	Deletes the current line of information.
;§	SPACE 1	FCTN F	Types the left brace (;§).
§;	SPACE 2	FCTN G	Types the right brace (§;).
[	SPACE 9	FCTN R	Types the left bracket ([).
]	SPACE 0	FCTN T	Types the right bracket (]).
<etx/eof>	SHIFT C	CTRL C	Indicates the end of a file.
<esc>	SPACE .	CTRL .	Tells the program to ignore previous text.
<tab>	SHIFT A	CTRL I	Moves the cursor to the next tab.
<return>	ENTER	ENTER	Tells the computer to accept the information you type.
<up-arrow>	SHIFT E	FCTN E	Moves the cursor up one line.
<right-arrow>	SHIFT D	FCTN D	Moves the cursor to the right one character.
<down-arrow>	SHIFT X	FCTN X	Moves the cursor down one line.
<left arrow> or <backspace>	SHIFT S	FCTN S	Moves the cursor to the left one character.

SECTION 2: GETTING STARTED

The steps for setting up the system and entering, saving, and running a TI PILOT program are included in this section. Please read this material completely before proceeding.

Use your Disk Manager Solid State Cartridge or the UCSD p-System Filer to make a back-up copy of the diskette which contains the TI PILOT interpreter (See Appendix G). Use this copy only for your own use. The original should be kept in a safe place. Initialize one or more blank diskettes for your program storage.

2.1 Setting Up the System

1. Be sure that all peripherals are attached to the computer and turned on. Refer to the appropriate owner's manuals for operation details.

2. If you want to create or modify a program, insert the Editor/Filer/Utilities diskette into a disk drive.

   If you want to run an existing program, insert the TI PILOT diskette into the first disk drive.

3. Turn on the computer console. After a short period of initialization, the p-System promptline appears.

NOTE: If you turn on the computer before inserting a diskette in a disk drive, you must insert a diskette and then press I to initialize the System before you can proceed.

4. If you want to create or modify a program, see "Entering and Saving a Program."

   If you want to run an existing program, see "Running a Program."

2.2 Entering and Saving a Program

1. After setting up the system, press E for E(dit to load the Editor.

2. Refer to the UCSD p-System Editor manual for detailed directions on entering a program. When you have completed your program, press Q for Q(uit. Then press W for W(rite.

3. If you haven't already done so, insert the diskette on which you wish to save the program.

4. Enter the filename for the program and press <return>.

5. In response to the message

   E(xit or R(eturn to the Editor?

   enter E to return to the p-System promptline.

2.3 Running a Program

1. If you have only one disk drive and you turned on the computer with a diskette other than the TI PILOT diskette in the drive, insert the TI PILOT diskette in the drive and select the I(nitialize command by pressing I. After reinitializing, the p-System promptline is displayed again.

2. In response to the p-System promptline, select the X(ecute command by pressing X. The computer then displays the following prompt:

   What file?

   Type the filename as

   #n:PILOT

   where n is the unit number of the disk drive containing the TI PILOT Interpreter diskette and PILOT is the filename. Then press <return>. The following chart lists the available disk drive unit numbers.

Device	Unit Number
First disk drive	#4
Second disk drive	#5
Third disk drive	#9

   See the UCSD p-System Filer manual for details concerning unit names and numbers.

NOTE: If problems occur, use the V(olumes command in the Filer program to verify which disk drives are currently being recognized by the system. If not all are being utilized, insert a p-System volume into each drive and use the I(nitialize command to restart the system.

3. The screen displays the message

   PATTERN FILE (PATTERNS):

   Type the filename of the file containing any special characters the program uses and press <return>. If there is no such file, or if the filename is PATTERNS, just press <return>.

NOTE: To use the default file PATTERNS, this file must be in unit #4 or be on the volume set as the prefix volume. A prefix volume is the first unit the system reads from when a filename does not contain a unit number or volume name. If a prefix volume has not been set, the system uses unit #4 as the prefix volume. The prefix volume can be set with the Filer P(refix command and can be verified with the V(olumes command.

The screen displays the message

COURSE:

Place the diskette that contains the program in a disk drive. Enter the disk drive number, a colon (:), and the name of the program. For example, if the name of the program is TEST and it is on a diskette in Disk Drive 1 (unit #4), enter

#4:TEST

and the computer runs the program.

5. When the program finishes running, the p-System promptline reappears.

2.4 Running the Demonstration Program

As an example of the procedure for running a TI PILOT program, you can run a program already on the TI PILOT diskette.

1. Be sure that all peripherals are attached to the computer and turned on. Refer to the appropriate owner's manuals for product details and the P-Code peripheral manual for p-System operation details.

2. Place the TI PILOT diskette in the first disk drive.

3. Turn on the computer console. After a short period of initialization, the p-System promptline appears.

4. Select the X(ecute command by pressing X. The computer then displays the following prompt:

   What file?

   Enter the following.

   #4:PILOT

   where 4 is the unit number of the disk drive containing the TI PILOT diskette and PILOT is the filename.

NOTE: Once the PILOT interpreter file has been loaded, you do not have to reload this file again (with the X)ecute command) each time you want to run a TI PILOT program. As long as no other programs have been loaded or executed, you can restart PILOT by pressing U (for U)ser Restart) when the p-System promptline is displayed.

5. When the screen displays the message

   PATTERN FILE (PATTERNS):

   press <return>.

6. When the screen displays the message

   COURSE:

   Type the following.

   DEMO

   and the computer runs the program. The name DEMO stands for DEMOnstration. (It is not necessary to precede the filename DEMO with a unit number if the p-System was initialized with the PILOT diskette in unit #4.)

Respond to the program as it directs you. Practice activities are included in section 3.

SECTION 3: TI PILOT TUTORIAL

This section assumes that you have had limited experience with the TI Home Computer, the p-System, or with computer programming. It is not intended to be a comprehensive course on TI PILOT or computer programming; rather it is intended to acquaint a newcomer to TI PILOT with many of the TI PILOT instructions and to give a sampling of how a program might be developed. If you have little previous programming experience, begin working in this section, consulting the other manuals for set-up instructions.

It is recommended that you study all of the manuals that accompany your computer equipment and software. These include manuals for the Peripheral Expansion System, the Disk Memory System, the P-Code Peripheral Card, and the Editor/Filer/Utilities program diskette.

If you are familiar with the UCSD p-System, have read all of the above-mentioned manuals, and are proficient at computer programming in a language other than PILOT (such as BASIC or Pascal), you may want to proceed to the reference section of this manual.

3.1 Demonstration Program

The demonstration program enables you to sample several different program sections within one program. This can give you an idea of the types of programs you might develop in TI PILOT. After choosing DEMO as the course (see the procedures outlined in section 2.4, "Running the Demonstration Program"), type your first name, press <return>, and read the explanation on the screen. When you are ready to continue, type Y (for Yes).

Then choose a program section (other then IMMED) from the menu, type GOTO and the section name (GOTO MATH, GOTO VERBS, etc.) depending on which program you select. Wait until section 3.2 (below) before you try the Immediate Mode program (IMMED). Follow the directions on the screen and answer the questions or problems, if any, as they are presented. When the program ends, you are returned to the menu.

You can leave the DEMO program by typing GOTO END and pressing <return>.

3.2 PILOT Immediate Mode

Note that the first program section listed on the DEMOnstration program menu is IMMED, which stands for Immediate Mode. Type GOTO IMMED and press <return>. In the Immediate Mode, the computer immediately performs the instructions you give to it. In this section, we will experiment with several of the instructions in the TI PILOT programming language.

There are seventeen instructions in TI PILOT. Each instruction has an abbreviation, which is called an op-code (operation code). An instruction op-code must be in capital letters. For convenience in this exercise, put the ALPHA LOCK key in the locked (down) position. (This key is at the lower left-hand corner of the TI-99/4A keyboard; press SPACE-4 on the TI-99/4 keyboard.) This causes all letters typed on the keyboard to be capital letters but does not affect the number or punctuation keys.

When an instruction op-code is given to the computer, it is always immediately followed by a colon (:).

3.1.1 The Type Instruction

Enter the following (type the line, correcting any mistakes with <left-arrow>, and then press <return>).

```
T: THIS IS A MESSAGE
```

The "T" is the op-code for the Type instruction. It tells the computer to display on the screen whatever follows the colon. (This is similar to a PRINT or DISPLAY command in other programming languages.) This displayed text is called the text-field. The space after the colon is optional; however, it is used throughout this manual to show the division between instruction op-code and text-field.

Notice that the text-field is displayed exactly as it was typed. The text field can consist of any kind of character, including both upper- and lower-case letters, numbers, punctuation, spaces, or other symbols (up to 80 characters in one text-field).

Experiment by entering several different Type instructions, using different kinds of characters. Use <left-arrow> to backspace and correct mistakes. Be sure to press <return> after each instruction to view the resulting display.

Though only forty characters per line are visible at any one time, the p-System allows for an eighty-column display. To see how this works, enter the following instruction:

```
T: NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY.
```

Notice that as you type more than forty characters on one line, the screen shifts to the next "window". The full eighty-column width is divided into three overlapping windows of 40 columns each.

```
                    <-----------2ND WINDOW---------->

<-----------1ST WINDOW---------->
                                  <-----------3RD WINDOW----------->

<-------------------------FULL 80-COLUMN LINE WIDTH------------------------>
```

After pressing <return>, you can view the displayed text by using ECIN_Z and ECIN_B (<screen left> and <screen right>, respectively). These keys allow you to use the three display windows. With the first window on the screen, press <screen right> once to view the second window and once more to view the third window. Then press <screen left> to go back to the second window and once more to return to the first window. Experiment with these keys until you feel comfortable with this feature.

3.1.2 Variables and the Compute Instruction

A variable is a character that is assigned a particular value. For example, if X=4, then the variable X has the numeric value of 4. In TI PILOT, the Compute instruction is used to assign numeric values to variables. The op-code for compute is C.

Enter the following instructions (remember to press <return> after each line). The comments on the right help to explain each instruction.

```
C: X=4          Assigns the value of 4 to the variable X.
T: X            Tells the computer to display the character "X"
                on the screen.
```

What happens? The computer doesn't perform anything on the screen after the Compute instruction, but it now holds the value for the variable X in memory. When the Type instruction is entered, only the character "X" is displayed. Now enter this instruction.

```
T: #X           Tells the computer to display the numeric value
                of X.
```

Notice that 4, the value of X, is displayed. Therefore, the # sign stands for "the current value of." Now enter this instruction.

T: THE ANSWER IS #X	Tells the computer to display "THE ANSWER IS" and the value of X. (Don't use a period.)

If you want to use a period at the end of this statement, type a space after the character X and then a period.

The variables studied above are numeric variables because the values assigned to the variables are numbers. Letters, words, or whole lines of text can also be assigned to variables. A set of characters assigned to a variable is called a "character string." Therefore, this type of variable is called a string variable.

3.1.3 The Dimension Instruction

A string variable in TI PILOT always consists of one alphabetical character (letter) and a dollar sign ($). A$, H$, and Y$ are all string variables. Before you can assign a value to a string variable, however, you have to tell the computer how many characters are in the string, so that it will reserve that many spaces in memory. Do this with a Dimension instruction (D:). Enter the following instructions.

D: M$(3)	Tells the computer to save three spaces for the string.
C: M$="DOG"	Assigns the character string "DOG" to the variable M$.
T: M$	Tells the computer to display the characters "M$".
T: M	Tells the computer to display the value of character string M$.

In the D: instruction, notice that the dimension (size) of the string is in parentheses. In the C: instruction, notice that the string, "DOG", is in quotation marks. In the first T: instruction, the computer merely displays the characters "M$". In the next T: instruction, however, notice that the string variable, M$, has an dollar sign in front of it. The $ sign functions as the # sign did with numeric variables, meaning "the current value of." Now enter the following instruction:

T: THE M CHASED THE CAT.

Notice that there is an extra space after the string variable M$ so that a space will be displayed between the words DOG and CAT.

In the Immediate Mode there is a convenient way to dimension three string variables with one instruction. If you enter the instruction.

U: ABC

the string variables A$, B$, and C$ are each set to the size of 256. This instruction is a Use instruction, which calls a subroutine in the Immediate Mode program.

3.1.4 The Graphics Instruction

The Graphics instruction (G:) allows you to use colors and moving pictures (sprites) in your programs. Enter the following instruction:

```
G: F12;87
```

The characters on the screen turn dark green. You instructed the computer to change the foreground color (with the F command) to color code 12 (dark green). Normally the characters on the screen (foreground) are black (color code 1) and the rest of the screen (background) is cyan (a pale blue, color code 7).

Now enter the following instruction:

```
G: B15
```

This time you have altered the background color (with the B command) to color code 15 (white).

You can change the color of both foreground and background in a single Graphics instruction. To do this, type a semicolon between the commands. Enter the following.

```
G: F15;B4
```

This time you changed the foreground to white and the background to a dark blue.

Below is a list of the sixteen colors available with the TI Home Computer and the TI PILOT code numbers for each. This same list is in Appendix D in the back of this book.

Color	Code #	Color	Code #
Transparent	0	Medium Red	8
Black	1	Light Red	9
Medium Green	2	Dark Yellow	10
Light Green	3	Light Yellow	11
Dark Blue	4	Dark Green	12
Light Blue	5	Magenta	13
Dark Red	6	Gray	14
Cyan	7	White	15

Note: Though these are the same colors used in TI BASIC, the code numbers range from 0 through 15 in TI PILOT. The code numbers in TI BASIC range from 1 through 16.

Some foreground and background color combinations are more readable than others. Enter the folowing instructions and observe the color changes when each one is performed. Then choose your own colors from the list above (and from Appendix E) and experiment with your own Graphics instructions.

```
G: F6;B11
G: F11;B4
G: F2;B15
G: F1;B10
G: F4;B15
G: F1;B13
G: F6;B14
G: F12;B11
G: F13;B15
```

You have probably noticed that when the screen is filled with instruction lines, the lines "scroll" from the bottom. To clear the screen you can use the following instruction.

```
G: T
```

Try it. The T command erases the screen and sets the screen to text mode (the mode the computer is already in). The cursor is returned to home position (upper-left-hand corner of the screen).

There are many other commands you can use with the Graphics instruction. Many of these involve sprites, which enable you to incorporate moving graphics into your programs for visual rewards for correct answers, illustrations, or simply something interesting to hold the students' attention while they are learning.

Enter the following Graphics instruction, making sure that the semicolons and commas are in the right places):

```
G: P;X1;S5,42,5,100,100,40,0
```

You should see a large asterisk or star moving across your screen from left to right. As in the Graphics instructions with the foreground and background commands, there is a semicolon between commands. The options or parameters within a command are separated with commas.

The P command erases the screen and sets the screen to pattern mode. In this mode, the display is 32 columns by 24 lines, resulting in larger characters on the screen. This mode also allows you to use sprites. The X1 command sets the sprite magnification to double, making the asterisk a larger character than it would normally be.

The remainder of the instruction selects the options concerning the sprite. The S command creates the sprite and defines it as sprite number 5. The number 42 defines the character pattern. 42 is the character code for the asterisk. (The codes for all of the predefined ASCII characters are listed in Appendix B in the back of this book.)

The next number, 5, selects the foreground color of the sprite, which is a light blue. The two 100s select the x-position and y-position, respectively. (Each character is generated by an 8 X 8 dot matrix, resulting in 255 possible positions for either x- or y-positions, though any y-position over 191 is below the range of the screen.) This sets the position of the sprite on the screen. The number 40 sets the x-velocity, or horizontal speed, of the sprite. The last number, 0, sets the y-velocity, or vertical speed, of the sprite.

X-velocity and y-velocity can be from -128 through 127. A value close to zero is very slow. A value far from zero is very fast. As you can see, when the sprite comes to the edge of the screen, it disappears and reappears in the corresponding position on the other side of the screen.

A positive x-velocity (as in this case) moves the sprite to the right and a negative value moves it to the left. A positive y-velocity moves the sprite down and a negative y-velocity moves it up.

In this case, since the y-velocity is 0, the sprite has no vertical movement. If both x-velocity and y-velocity are zero, the sprite is stationary. If both x-velocity and y-velocity are non-zero, the sprite moves at an angle in a direction determined by the actual values.

Try changing the options of this Graphics instruction. Enter the instruction as before, first changing the foreground color of the sprite. (Use another number of your choice from the color code chart rather than the number 5.) Then try changing the character pattern from an asterisk to another character. (For example, use a number 36, 35, or 33 rather than the number 42. Consult the ASCII Character Codes chart in Appendix B.)

Then try changing the x- and y-velocities, using the information given in the last few paragraphs. Try several different combinations of x- and y-velocities.

When you are finished experimenting with a sprite, use the halt-sprite command, which is H followed by the sprite number. NOTE: If you do not use this command, your system will be slowed down. To return to the text mode (normal size characters), use the T command. Enter these commands with this instruction.

```
G: T;H5
```

3.1.5 The Sound Instruction

The Sound instruction (S:) enables you to send audio signals to the student in the form of beeps, tones, or music.

Enter the following instruction.

```
S: 1,C;V10;T262,10;P
```

This instruction tells the computer to process the given sound list. The TI Home Computer has four "voices," that is, three tone generators and one noise generator. The 1 in the above instruction selects voice 1. The C command clears the sound list for voice 1. The V command sets the volume at 10 (on a scale of 0 to 15). The T command selects a tone with a pitch of 262 cycles per second (middle C on a piano) with a duration of 10 beats. The P command ends the sound list and tells the computer to play the sound list. To begin a new sound list the C command must be used to clear or erase the information from the old sound list.

Now enter this instruction.

```
S: 1,C;V10;T262,10;T523,10;T131,10;P
```

This is the same instruction except two additional tones are used. First you hear middle C, then the C an octave below, and then the C an octave above middle C.

Now enter the same instruction except add S50; before the P command. It will look like this.

```
S: 1,C;V10;T262,10;T523,10;T131,10;S50;P
```

Notice how the speed of the tones played is affected by the S command. This command alters the tempo of a series of notes. The number immediately following the character S (the tempo selected) can range from 1 to 32767.

Experiment with the Sound instruction by using different volumes, pitches, durations, and tempos.

3.1.6 The Speech Instruction

NOTE: The Solid State Speech™ Synthesizer is required for this instruction to function.

With the Speech Instruction you can send verbal directions or reinforcement to the student. The computer speaks the string of words contained in the string variable following the V: instruction. Space for the string variable must be reserved (by a D: instruction) and the variable must be defined (by a C: instruction) for the V: instruction to work. Enter the instructions below and listen to the result.

```
D: F$(80)
C: F$="I AM A COMPUTER"
V: F$                          Computer speaks the words contained
                               in the string variable F$.
```

Any of the 373 words and phrases included in the Speech Synthesizer's resident vocabulary can be spoken by the computer. A list of this vocabulary is in the booklet which accompanies the Speech Synthesizer. Experiment by redefining F$ to be a different phrase (with another C: instruction) and making the computer speak it (with another V: Instruction).

These are the TI PILOT instructions you have studied thus far:

T: Type
C: Compute
D: Dimension
G: Graphics
S: Sound
V: Speech

To return to the menu from the Immediate Mode, type GOTO MENU and press <return>. To leave the DEMO program and proceed to section 3.3, type GOTO END and press <return>.

3.3 Pattern Editor

The Pattern Editor feature of TI PILOT offers an easy way to create and define your own characters or patterns and then store them in a file for later use. With the Pattern Editor you can create your own font for the alphabet, create special alphabetical characters for a language other than English, or create special graphics characters for your use with graphics applications.

A video screen is divided into thousands of tiny dots called pixels. In Text Mode, each printing position on the screen is made up of 48 of these dots (an 8 X 6 block); in Patterns Mode, each printing position on the screen is made up of 64 of these dots (an 8 X 8 block). The computer displays a character or pattern on the screen by turning the dots that are a part of the character "on" and turning the other dots within that printing position "off." Specifically, the dots that are turned "on" are the foreground color, and the dots that are turned "off" are the background color.

Your diskette contains two identical files, both of which contain the standard p-System patterns. The filenames for these files are PATTERNS and PATTERNCOPY. The Pattern Editor enables you to change existing characters or add characters to the PATTERNCOPY file.* You can also create other pattern files.

To load the Pattern Editor program into your computer, place the TI PILOT diskette into Disk Drive 1 (unit #4). Then, with the standard p-System promptline at the top of the screen, do the following.

Screen Display:	You:
1. Command: E(dit, R(un, [p-System promptline]	1. Press X)ecute.
2. What file?	2. Enter: #4:PATED
3. Initializing...(dark blue screen) Name of pattern file (PATTERNS):	3. Enter: PATTERNCOPY
4. Reading pattern file...	

WARNING: Always load the PATTERNCOPY file, rather than the PATTERNS file, when making changes to existing patterns. Otherwise, you will change (and thereby lose) the predefined characters used for program display.

[Display—pattern mode]

[8 X 8 grid with blinking cursor]

```
  1 2 3 4 5 6 7 8
1>_>_>_>_>_>_>_>_>1
2>_>_>_>_>_>_>_>_>2
3>_>_>_>_>_>_>_>_>3
4>_>_>_>_>_>_>_>_>4
5>_>_>_>_>_>_>_>_>5
6>_>_>_>_>_>_>_>_>6
7>_>_>_>_>_>_>_>_>7
8>_>_>_>_>_>_>_>_>8
```

PATTERN

SINGLE

E,S,D,X: Draw
e,s,d,x: Move cursor
<space> toggle
V)save pattern
P)pattern mode
U)remove pattern
Q)quit
G)get pattern
T)text mode
?)next menu

(next menu)
F) change foreground color
B) change background color
1) define single character
4) define quad character
K) clear pattern
L) list patterns
I) invert pattern

On the screen you see an 8 X 8 grid (8 rows and 8 columns) with the cursor at the top-left corner. This grid shows, greatly enlarged, one printing position in Pattern Mode. The row numbers are at the sides of the grid and the column numbers are at the top of the grid. Each square in the grid represents one tiny dot on the video screen. At the bottom of the screen you have a "menu," or selection list of available options. Notice that the menu lists T for Text Mode. Press T.

Now an 8 X 6 grid (8 rows and 6 columns) appears with the same menu at the bottom of the screen. This grid shows, greatly enlarged, one printing position in Text Mode. Again, there are row numbers and column numbers at the top and sides of the grid. Now press P to return to the Pattern Mode screen.

As displayed on the menu, the arrow keys (E, S, D, and X) are used to move the cursor. If the arrow keys are in upper-case mode, the cursor will draw a line as you move it. If the arrow keys are in lower-case mode, they will move the cursor without drawing a line. If a pattern is currently displayed, the movement of the cursor in upper-case mode will "toggle" that position; that is, a colored square will become blank and vice versa.

With the arrow keys in lower case (ALPHA LOCK in the up position), move the cursor around the grid by using the arrow keys. The SPACE BAR is a toggle switch between upper- and lower-case. If you press the SPACE BAR once, you change to upper-case. Remember that you can advance the cursor one square at a time or hold the key down longer than one second to automatically move the cursor. Notice also that the pattern you are creating is displayed at the top right of the screen in its normal size.

You can use the G(get pattern command to view some of the patterns stored in the PATTERNCOPY file. Press T for text mode, and then press G to get a pattern. Enter 32 as the ASCII code for the space character. Your grid becomes blank. You can get pattern 32 anytime you want to clear or erase the grid. Now get patterns 36, 47, 65, and 77 (by pressing G and entering the ASCII code number) to view the $, /, A, and M characters. Notice that the ASCII code number of the currently displayed pattern is displayed on the right side of the screen.

You can experiment with any of the existing patterns by changing them on the grid. You will not alter any of the existing patterns in the PAT file unless you S)ave the altered pattern to the same code number, which will replace the old pattern with the new.

The easiest method for developing a pattern is to leave the arrow keys in lower case and simply use the SPACE BAR to add a dot to the pattern. Let's make a checkerboard pattern using this method. First, make sure the keyboard is in lower case mode. (Put ALPHA LOCK in the up position if you are using the TI-99/4A; press SPACE 4 if you are using the TI-99/4).

Get pattern 32 to clear the grid. Begin with the cursor in the upper-left-hand corner of the grid. Press DOWN_ARROW (the X key) to move the cursor one square down. Press the SPACE_BAR to color in that square (row two, column one). Now press DOWN_ARROW twice and the SPACE_BAR again to color in that square (row four, column one). Continue this sequence until the cursor reaches the bottom of column one.

Now press RIGHT_ARROW once and UP_ARROW (the E key) once. Press the SPACE_BAR to color in the square at row 7, column 2. Press UP_ARROW twice and press the SPACE_BAR to color in the square at row 5, column 2. Continue this sequence until the cursor reaches the top of column 2. Now continue this procedure until you have moved the cursor up or down each column to create an alternating pattern of "on" and "off" squares that resemble a checkerboard. If you make a mistake, position the cursor to the point where the error was made and use the SPACE_BAR to "toggle" or reverse the current status of that square (a colored square will become blank and vice versa).

Now that you have created a pattern, you may want to save it. If you don't know what pattern number to use, press L to list all the patterns currently defined in the file. You can choose an unused pattern number or replace a pattern already in the file. Then press V (save pattern) and enter the pattern number (from 0 through 255). If you decide not to save the pattern you can enter 256 as the pattern number, and you will leave the V command without anything being saved.

Press Q to put the pattern generator in Quad mode. The screen now displays a 16 X 16 grid and QUAD instead of SINGLE to denote the mode you are in. Press G and enter 65 to get pattern 65 (A). Now the letters A, B, C, and D are displayed on the grid. Note the order in which the characters are displayed on the grid. This is the order in which quad characters are stored. The 8 X 8 upper-left-hand block displays the pattern number you designate. The next three consecutive pattern numbers are automatically assigned to the remaining three 8 X 8 blocks. The order that they are assigned is always the same as in the ABCD example.

The K command will clear the display from the grid just as getting pattern number 32 did. The U command will remove a pattern which you designate by number from the file. Any patterns saved or removed in Quad mode will also affect the three consecutive patterns following the one you specify.

The F and B commands change the foreground (on) color and background (off) color for a pattern. These commands are only for display purposes while in the Pattern Editor. To display patterns with altered colors in a program, you must use the Graphics instruction (G: Cn,f,b). The I command inverts the character or pattern displayed.

To quit the Pattern Editor, press Q (as indicated on the menu). The following prompt appears on the screen.

```
Update pattern file (Y/N):
```

You can press Y if you want to update a pattern file with the patterns that you have saved. If you press Y, the next message prompts for the filename of the file you want to update.

```
Pattern file name (PATTERNS):
```

At this point, you can simply press <return> to update the default file (PATTERNS), or you can enter a new file name for a special patterns file.

If you do not want to update a pattern file, press N at the first prompt. At this point, you can return to the Pattern Editor (press Y) or go to the main p-System promptline (press N).

```
Return to editor (y,n)
```

Press N to continue to the next section.

3.4 Simple Programming

A computer program is simply an ordered list of instructions that you tell the computer to perform. Unlike in the Immediate Mode, the instructions are not performed until you tell the computer to perform them. With the Editor and Filer programs which are on diskette, you can change your program, save it for future use, and perform many other operations.

If you have at least two disk drives, insert the Editor/Filer diskette into Disk Drive 2 and press I to reinitialize the system.

If you have only one disk drive, remove your PILOT diskette and insert the Editor diskette. You will need to insert the Editor/Filer diskette each time you want to perform an Edit or File operation. Then you will remove that diskette and re-insert the PILOT diskette, which contains your workfile.

For this exercise, we will assume you have at least two disk drives. Press E for Edit. When this promptline

```
No workfile, File (<ret> for none)?
```

appears, press <return> to verify that you do not have a current workfile.

The Editor creates a workfile for you. The Edit promptline at the top of the screen lists some of the options available in the Edit program. Press I to Insert. Notice that the promptline at the top of the screen changes. Now you are ready to enter a program. If you make a mistake, use <left arrow> to backspace and type over it as you did before.

3.4.1 The Accept and Match Instructions--The Y and N Conditioners

Enter the following instructions:

T: WHAT IS 8 X 8?	Displays question to student.
A:	Accepts student's response.
M: 64	Compares student's response with correct answer to see if they match.
TY: THAT'S RIGHT!	If answers match, this line is displayed.
TN: NO, TRY AGAIN.	If answers do not match, this line is displayed.

Two new instructions are used in this program. The Accept instruction (A:) receives a student's reply. The Match instruction (M:) compares the student's reply with the correct answer (in the text-field of the Match instruction).

Notice that the last two lines are simply Type instructions with conditioners. The Y (Yes) conditioner causes the TY: line to be displayed if the student's reply does match the correct answer. TN: is a Type instruction with an N (No) conditioner. If the student's reply does not match the correct answer, this line is displayed.

Check the program you have entered for errors. If you notice an error, you can still correct it by using <left-arrow>, even if you have already entered the line. Just hold down <left-arrow> and the cursor will backspace to the beginning of one line and then jump to the previous line. You can also use <esc> (SPACE_. on the TI-99/4 and CTRL_. on the TI-99/4A) to "escape"--to tell the computer to ignore the previous text. This is used if you press a wrong command or make several mistakes so that you want to erase and start over.

When you are sure there are no errors, press <ext/eof> (SHIFT_C on the TI-99/4, CTRL_C on the TI-99/4A). This marks the end of your file and takes you out of the Insert mode. Notice that the promptline at the top of the screen returns to the Edit options line. Press Q to Quit the Editor program. When the Quit menu appears, press U to Update the workfile and leave. The next screen gives you the p-System promptline and the length of your file in bytes (characters).

Now you can execute (run) your program just as you have already done with existing programs. Press X for X)ecute, enter #4:PILOT in response to "What file?", press <return> in response to PATTERN FILE (PATTERNS):, and enter SYSTEM.WRK for the course (program). This is the system workfile into which the system saved your program.

Pretend you are the student and respond to the question your program displays. Type 64, press <return>, and you see the appropriate response. The program ends and the p-System promptline appears.

You can restart the program by pressing U for U(ser restart. With this command you can quickly run a program that is already in memory without having the repeat the steps as above. U(ser restart must be the first p-System command used after a program is loaded into memory. Restart the program, and this time deliberately answer the question incorrectly. Note that you get a different response than on the previous program run.

You can now go back and edit your program to make it a little more elaborate. This time let's call the student by name. Press E to return to the Edit mode. Now your program is displayed with the cursor at the beginning of your program. Press I to insert these lines (be sure to press <return> at the end of each line):

```
D: A$(10)                          Saves space for ten characters for
                                   string variable A$.
T: ENTER YOUR FIRST NAME, PLEASE.  Displays instructions to student.
A: $A$                             Accepts the student's response and
                                   assigns it to A$.
```

After you have corrected all errors, press <ext/eof> to enter these lines. As you can see, you have defined the dimensions for the string variable, displayed the instructions to the student, and told the computer to accept the student's name and assign the name to the variable $A.

Now press <right-arrow> until the cursor is positioned over the "W" in the word "What." Press I to insert the following:

```
$A$ ,                          (Include a space after the comma.)
```

Be sure to insert the spaces before and after the comma. Press <ext/eof>. The line should now look like this:

```
T: $A$ , WHAT IS 8 X 8?
```

If the line on your screen is not the same as the line above, use <left-arrow> to backspace to the error and follow the Insert procedure described above to correct the error. NOTE: Refer to the p-System Editor/Filer/Utilities manuals for procedures for other command options.

Now move the cursor to the TY: line and position it over the exclamation point (!). Press I and enter:

```
, $A$                              (Include a space after the last $.)
```

Be sure to type a space after the string variable. Press <ext/eof> and the line should look like this:

```
TY: THAT'S RIGHT, $A$ !
```

Insert corrections if necessary. Now move the cursor to the next line (TN:) and insert the same thing, starting with the cursor over the period. Enter:

```
, $A$                              (Include a space after the last $.)
```

Press <ext/eof> and the line should look like this:

```
TN: NO, TRY AGAIN, $A$ .
```

Insert any corrections if necessary. The entire program should now look like this:

`D: A$(10)`	Saves space for ten characters for string variable A$.
`T: ENTER YOUR FIRST NAME, PLEASE.`	Displays instructions to student.
`A: $A$`	Accepts the student's response and assigns it to A$.
`T: $A$ , WHAT IS 8 X 8?`	Calling the student by name, displays question to student.
`A:`	Accepts student's response.
`M: 64`	Compares student's response with correct answer to see if they match.
`TY: THAT'S RIGHT, $A$ !`	If answers match, this line is displayed (calls student by name).
`TN: NO, TRY AGAIN, $A$ .`	If answers do not match, this line is displayed (calls student by name).

Now X)ecute the program following the steps as before. Notice the difference in this version of the program.

3.4.2 The Jump Instruction

In the program as developed so far, the program ends, regardless of which answer the student gives. If the student answers the problem incorrectly, he or she doesn't have a chance to "TRY AGAIN."

We can use the Jump instruction to cause the computer to transfer to an earlier or later line in the program. Since we want the student to have another chance to answer the question, let's jump back to the Accept instruction. Press E for E(dit, and then use the <down-arrow> to move the cursor down past the last line of the program. Press I for I(nsert, and then enter the following instruction:

```
J: @A
```

This instruction will cause the computer to jump back to the last Accept instruction. However, you wouldn't want it to jump back if the question were answered correctly. Therefore, let's insert another jump instruction. Position the cursor just after the TY: instruction and insert:

```
JY: CONTINUE
```

Press <ext/eof> and position the cursor at the end of the program, just after the J: instruction. Now enter the following:

```
*CONTINUE
```

Your entire program should now look like this:

```
D: A$(10)
T: ENTER YOUR FIRST NAME, PLEASE.
A: $A$
T: $A$ , WHAT IS 8 X 8?
A:
M: 64
TY: THAT'S RIGHT, $A$ !
JY: CONTINUE
TN: NO, TRY AGAIN, $A$ .
J: @A
*CONTINUE
```

Press <ext/eof>, press Q, and press U. Then X)ecute the system workfile. When the program runs, deliberately answer incorrectly a few times. What happens?

Now the program keeps running as long as the answer is incorrect, always giving the student another try. When the correct answer is given the program ends. The last line of the program, *CONTINUE, is not an instruction. It is called a label. Notice that it is marked by an asterisk (*). Notice also that the JY: instruction (with a Y conditioner) has the word "CONTINUE" after it. This instruction tells the computer to jump to the label called CONTINUE if the answer is correct. (The computer only recognizes the first six characters in a label; therefore, it matches the letters "CONTIN" and ignores the letters "U" and "E".)

3.4.3 The Remark Instruction

The Remark instruction (R:) can be used to to document your programs. The computer does not read this instruction. Remarks serve as a reminder to yourself or to someone else who may study or further develop your program. Press E to return to Edit mode and Insert the following instruction at the beginning of the program:

```
R: A PRACTICE PROGRAM IN TI PILOT
```

Press <ext/eof>. You can now save this program under a new filename. To do this, press Q(uit and then press W(rite. The following promptline appears:

```
>Quit:
'$'<ret> writes to PILOT:SYSTEM.WRK.TEXT
Output file (<cr> to return)
```

Type the new filename and press <return>. You can make up your own filename or simply call it PRACTICE.TEXT. Remember to always include the suffix .TEXT on your source files to identify them as text files created by the Editor.

If you leave the Editor after writing to another file, the SYSTEM.WRK file is not updated and remains as it was before the edit session. If you want to also update the SYSTEM.WRK file, return to the editor, Q)uit again, and choose the U)pdate option.

To clear or erase the SYSTEM.WRK file, execute the F)iler and use the N)ew command.

This concludes the practice exercises in section 3. You are now ready for a more detailed study of TI PILOT. In this section you have studied and experimented with variables, programs, labels, and many of the instructions in TI PILOT. These are the TI PILOT instructions you have studied thus far:

T: Type
C: Compute
D: Dimension
G: Graphics
S: Sound
V: Speech
M: Match
A: Accept
J: Jump
R: Remark

SECTION 4: THE FORMAT OF A TI PILOT INSTRUCTION

This section is an overview of the format of a TI PILOT instruction, including an overview of the kinds of data used by the instructions.

A TI PILOT program consists of a set or series of instructions. All of the instructions in a program either define data or cause an operation to be performed.

An instruction in a TI PILOT program consists of a single line of up to 80 characters. An instruction can include six kinds of elements. These elements are

- Label
- Op-code
- Modifiers
- Conditioners
- Relational expression
- Text-field

Not all of these fields must be present in an instruction. However, when the fields are present, they must be in the order listed above.

4.1 Labels

The label field is optional. It is used when you want to name a specific, individual instruction in a program. You may use a label alone on a line; in that case, the label is associated with the following instruction.

The computer recognizes only the first six characters of a label. When used in a label field, the label begins in column two, preceded by an asterisk in column one. When a label is used in a text field, it is not preceded by an asterisk. The first of these characters must be a letter; the other characters may be letters or numbers. There can be up to twenty-five labels in a TI PILOT program.

If a label is followed by an op-code on the same line, there must be at least one space between the last character of the label and the first character of the op-code. There must not be any spaces between the asterisk and any of the characters in the label. The characters of a label constitute a name which is associated with the instruction (or set of instructions) following it.

You can choose any name you want for a label. Your programs are easier to understand, however, if you choose names which reflect the purpose of the program segment beginning at the label. Because the computer doesn't recognize anything past the sixth character in a label, you can use the space thereafter to document your program as you would with the Remark instruction.

For example, the label

```
*INTRO
```

may be used for the introductory segment of a program, and the label

```
*ANALYS
```

may be used for the program section which performs an analysis of test scores.

Here are some examples of correct and incorrect labels as used in the label-field. Assume that the asterisk preceding each label begins in the first column of a line.

*GOOD	Correct.
*EIGHTY	Correct.
*A73	Correct.
*A73M5	Correct.
INTRO	Incorrect; no asterisk in front of the name.
*INTRODUCTION	Correct (computer only recognizes first six characters).
* INTRO	Incorrect; a space between the asterisk and the label.
*IN TRO	Incorrect; a space within the label.
*4PLUS2	Incorrect; first character after the asterisk is not a letter.
*FOUR+2	Incorrect; the plus sign is not a letter or a number.

Note: When used in the text-field of an instruction, a label is not preceded by an asterisk. For example, in the instruction

```
J: GOOD
```

J: is the op-code for the Jump instruction and GOOD is the label to which the Jump transfers control.

4.2 Operation Codes (Op-codes)

The operation code identifies the kind of instruction. The operation code (op-code) must appear first in an instruction unless the instruction contains a label. In that case, it follows the label. There must be at least one space between the last character of the label and the first character of the operation code.

The TI PILOT operation codes are discussed in section 5.

An operation code specifies the action an instruction is to perform. Three items can be appended to an operation code to alter its performance. These are a modifier, a conditioner, and a relational expression.

An op-code (plus any modifiers, conditioners, or relational expressions) is followed immediately by a colon (:).

For example, T: is the op-code for a type instruction.

4.3 Modifiers

A modifier is a single letter appended to an op-code which changes the way in which an instruction is performed. If used, a modifier follows the last letter of the op-code (before the colon).

For example, T: is the op-code for the Type instruction and in the instruction

```
TH: THAT'S RIGHT.
```

H is a modifier.

A modifier (if used) must come before any conditioner (if used).

Here is a list of the modifiers.

Modifier Code	Modification
H	Used with the Type operation code (TH:), suppresses the carriage return and line feed after a message is displayed.
X	Used with the Accept operation code (AX:), suppresses the automatic editing of a student response.
J	Used with the Match operation code (MJ:), causes an automatic jump with a correct match.
S	Used with the Match operation code (MS:), permits simple spelling errors.

Even though specific modifiers make sense only with certain op-codes, they can be combined with any op-code. A modifier is ignored if it is not applicable to the op-code.

4.4 Conditioners

A conditioner is a single character attached to an op-code (or modifier, if present) which determines whether or not the instruction is performed, depending upon the truth of the condition(s).

For example, T: is the op-code for the Type instruction and in the instruction

```
THY: THAT'S RIGHT.
```

Y is a conditioner. It causes the Type instruction to be performed only if the Y (Yes) conditioner is set.
Here is a list of the conditioners

Condition Code	Meaning
Y	Perform the instruction if the Y (Yes) conditioner is set.
N	Perform the instruction if the N (No) conditioner is set.
C	Perform the instruction if the last relational expression was true.
E	Perform the instruction if the E (Error) conditioner is true.
n	Digit conditioner; perform the instruction if the value of n is the same as the value in the Answer Counter (%A). The Answer Counter is described in sections 4.8 and 5.9.9.

Any conditioner must immediately follow a modifier, if present. Conditioners themselves may be used in any order. Any conditioner can be used with any op-code.

4.5 Relational Expressions

A relational expression is an expression enclosed in parentheses which either causes the instruction to be performed if the relational condition is true or causes the instruction to be skipped if the relational condition is false. A relational expression must follow any conditioners, if present.

The opening parenthesis of a relational expression must be adjacent to the op-code or the last modifier or conditioner, as the case may be. The expression within the parentheses may contain spaces.

For example, J: is the op-code for a Jump instruction and the instruction

```
J(Q+2=8): EXER17
```

is performed if the relational expression Q+2=8 is true. The J: instruction is not performed if the relational expression is false.

4.6 Text-Field

The text-field is required for certain instructions. The text-field is a group of characters to be used in an instruction's performance. The contents of the text-field depend on the operation of the instruction. The text-field must follow the operation code and any modifier, conditioner, or relational expression, if present. There can be spaces before the text-field and within the text-field.

4.7 Data Types

TI PILOT uses two kinds of data: numeric data and string data. Numeric data consist of numeric values. String data consist of printable characters. Within the computer, these two categories of data are represented differently. Numeric data are represented in a form for computation. String data are represented in a form for displaying.

4.7.1 Numeric Data Types

There are four types of numeric data: constants, variables, arrays, and functions.

4.7.1.1 Numeric Constants

Numeric constants, as the name implies, are numeric values which do not change. For example, 5, 173.89, and 3/15 - 2 are all numeric constants.

Numeric constants can be written in either a decimal format or a floating point format.

A decimal format is a list of decimal digits (0 through 9) and may include a decimal point and may be preceded by a plus (+) or minus (-) sign. For example, 8193 is a decimal format numeric constant. So is 81.93, -8.193, and +819.3.

A floating point format is a decimal number followed by the letter E, a sign (+ or -), and a one- or two-digit integer. The decimal number preceding the letter E is a mantissa. The integer value following the letter E is an exponent. A negative exponent is preceded by a minus sign (-). A positive exponent may be preceded by a plus sign (+) or the plus sign may be omitted. The exponent is the power of 10 which is multiplied by the mantissa to determine the value of the number. For example, 54E2 is a floating point numeric constant equal to 54 times 10^2 or 54 times 10 times 10 or 5400.

Here are some other examples of floating point numeric constants.

```
-89E3              (equals -89000)
8.9E3              (equals 8900)
89E-3              (equals 0.089)
8.9E-3             (equals 0.0089)
-89E-3             (equals -0.089)
-8.9E-3            (equals -0.0089)
```

4.7.1.2 Numeric Variables

Numeric variables are names given to numeric values. As as the name "variable" implies, these values can be changed by your program.

A numeric variable is designated by a single upper-case letter or an upper-case letter followed by a single digit. For example, N, X, N1, and X9 are all valid numeric variables. The following are not valid numeric variables: t (not an upper-case letter), T90 (more than one digit following the letter), 9T (the first character is not a letter).

7.1.3 Numeric Arrays

A numeric array is a set of values organized into a matrix with defined dimensions. An array is given a name and is defined by a Dimension instruction (see section 5.10 for a description of the Dimension instruction). For example, the Dimension instruction

```
D: B6(2,3)
```

defines an array named B6 with 2 rows and 3 columns.

The name of an array can be a single upper-case letter or an upper-case letter followed by a single digit (the same as a numeric variable name).

The name of the array is followed by a subscript consisting of either one or two numbers enclosed within parentheses. When the subscript has two numbers, the numbers are separated by a comma.

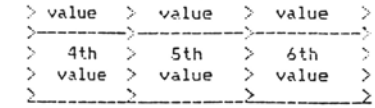
An array in TI PILOT can have one or two dimensions. The first number inside the parentheses is the number of rows. The second number (if present) is the number of columns.

The following are examples of numeric array designators.

```
N(2,3)
Z8(9)
```

The smallest permissible subscript number is zero and the largest is 255.

The 2 by 3 array looks like this.

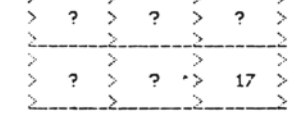


You can assign numbers to the elements of an array by means of a Compute instruction (see section 5.9 for a description of the Compute instruction).

For example,

```
C: B6(2,3) = 17
```

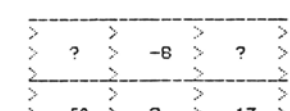
sets the element in the 2nd row and 3rd column to the value, 17.



Then, the two statements

```
C: B6(1,2) = -8
C: B6(2,1) = 50
```

cause the array to look like this.



A one-dimensional array has only one row, but can have more than one column.

4.7.1.4 Numeric Functions

A numeric function automatically performs a numerical calculation on a numeric value. The function is specified by a key word and the numeric value on which the calculation is performed is enclosed in parentheses following the key word.

For example, the instruction

```
C: X = ABS(Z)
```

calculates the absolute value of the numeric variable Z and assigns that value to the numeric variable X.

Section 8 describes the numeric functions available with TI PILOT.

4.7.2 String Data Types

A string consists of one or more printable characters. There are four types of string data: constants, variables, pseudo-variables, and string functions.

7.2.1 String Constants

A string constant is a string value which does not change. It must be enclosed in quotation marks, except in the text-field of a Type or Match instruction. For example, "XYZ" and "FETCH, SPOT!" are string constants.

A string constant can be used to assign a value to a string variable or it can be used in a string expression.

When a string constant is used in a Compute instruction, a pair of quotation marks indicates the beginning and end of the string itself. For example,

```
C: Z$ = "VWXYZ"
```

assigns the string value VWXYZ to the string variable Z$.

4.7.2.2 String Variables

A string variable is a name given to a string, the value of which can change. A string variable is designated by a single upper-case letter (or a letter and a single digit) followed by a dollar sign ($). For example, N$, X$, N1$, and X9$ are all valid string variables. The following are not valid string variables:

```
T          (no dollar sign)
t$         (not an upper case character)
T90$       (more than one digit following the letter)
9T$        (the first character is not a letter)
```

A string variable can be assigned a value by a Compute instruction. For example, the instruction below assigns the word TURTLE to the string variable A$.

```
C: A$ = "TURTLE"
```

A string variable cannot be used in a Compute instruction until space has been allocated for it by the Dimension (D:) instruction (see section 5.10 for a description of the Dimension instruction). The Dimension instruction establishes the maximum length for the string variable. Its length can be from 1 through 999 characters. For example, the instruction

```
D: X1$(80)
```

defines a string variable named X1$ with a maximum length of 80.

The characters in a string are stored from left to right. If the maximum string length established by the Dimension instruction is not large enough to contain the characters assigned to the string variable, characters are truncated (deleted) on the right. For example, the Dimension instruction

```
D: Z$(10)
```

assigns the string variable Z$ a maximum length of 10 characters, and the instruction

```
C: R$ = "PQRSTUVWXYZ"
```

causes the ten characters PQRSTUVWXY to be stored in R$. The Z is truncated.

At this point, the current length of R$ is equal to the maximum length (10).

The length of a string is changed whenever a different number of characters is stored in it. The instruction

```
C: R$ = "JKL"
```

causes the characters JKL to be stored in R$. The current length of R$ is 3.

4.7.2.3 String Pseudo-Variables

A special case of a string variable is a pseudo-variable, which identifies a portion or subset of a string. To define a pseudo-variable, use the form

```
A$(x,y)
```

where x gives the position of the character in the string which begins the pseudo-variable and y gives the number of characters.

For example,

```
A$(4,2)
```

refers to the 4th and 5th characters in the A$ string variable. If A$ = "TURTLE", then A$(4,2) refers to the string consisting of the characters T and L.

A pseudo-variable can be used in a Compute instruction. For example,

```
C: Z$ = "DOG & PIG"
C: Z$(7,3) = "CAT"
```

results in Z$ = DOG & CAT. (The characters C, A, and T replace the characters P, I, and G.)

If a second number is not specified in the subscript, a default value of 1 is used. The first character in a string is numbered 1. For example,

```
C: Z$ = "VWXYZ"
C: G8$ = Z$(2,3)
```

assigns the characters WXY to string variable G8$. Then, the instruction

```
C: F$ = G8$(2)
```

assigns the character X to the string variable F$.

The value assigned to the string variable is truncated on the right if necessary. For example, the Dimension instruction

```
D: D$(15)
```

assigns a maximum length of 15 to the string variable X$. Then the instruction

```
C: X$ = "HORSES AND COWS"
```

assigns HORSES AND COWS to X$.

The instruction

```
C: X$(12,6) = "CATTLE"
```

results in X$ = HORSES AND CATT.

The value assigned to a string variable may be padded on the right with spaces if necessary. For example, with X$ = HORSES AND COWS

```
C: X$(1,6) = "PIGS"
```

results in X$ = PIGS AND COWS (3 spaces between PIGS and AND).

The length of a string variable does not change as the result of using a pseudo-variable.

The number in a subscript can be specified as a numeric constant or a numeric variable. For example,

```
C: N = 7
C: Z$ = "RIKKI-TIKKI-TAVI"
C: G$ = Z$(N,5)
```

assigns the characters TIKKI to the string variable G$

4.7.2.4 String Functions

A string function performs an operation on a string automatically. The function is specified by a key word, and the name of the string on which the operation is performed is enclosed in parentheses after the key word.

For example, the instruction

```
C: Z$ = CHR(13)
```

automatically assigns the character code for a carriage return character to the string variable Z$.

Section 8.3 describes the string functions available with TI PILOT.

4.8 System Variables

There are two system variables which are used for special purposes in a TI PILOT program: the %A variable and the %B variable. The contents of these two variables are automatically updated by TI PILOT and can be accessed by appropriate instructions.

4.8.1 Answer Counter (%A)

The %A (Answer Counter) system variable is a numeric variable that contains the current value of the Answer Counter. The %A value can be used to cause an instruction to be performed after a given number of student responses.

The value is set to 1 the first time a specific Accept instruction is performed (see section 5.5 for a description of the Accept instruction). The value is automatically incremented by one each time the same Accept instruction is performed without an intervening Accept instruction. The %A value is the value automatically tested by the Digit conditioner.

A Compute instruction can include the system variable %A. For example,

```
C: N = %A + 2
```

assigns the value 7 to N if the value in %A is currently 5.

4.8.2 Answer Buffer (%B)

The %B (Answer Buffer) system variable is a string variable that contains the characters last entered by the student. A student's response can be saved by assigning %B to the contents of a string variable in the program.

A Compute instruction can include the system variable %B. For example,

```
C: S$ = %B(10,3)
```

assigns to string variable S$ the tenth, eleventh, and twelfth characters in the Answer Buffer.

SECTION 5: TI PILOT INSTRUCTIONS

TI PILOT is based on 16 instructions. Each instruction has an operation code or "op-code" which identifies the instruction and is a "nickname" for the instruction. Each of the instruction op-codes consists of one or two letters followed by a colon (:).

The TI PILOT instructions are listed in the following table in the order in which the instructions are introduced in this section.

Name	Operation Code	Meaning	Section
Problem	PR:	Marks the start of a problem group and sets options.	5.1
Remark	R:	Adds comments about the program from the author.	5.2
Type	T:	Displays text to a student.	5.3
Accept	A:	Receives a student's reply.	5.4
Match	M:	Compares a student's reply to an expected reply.	5.5
Jump	J:	Transfers program control to another segment of the program.	5.6
Use	U:	Calls a subroutine to perform an operation.	5.7
End	E:	Terminates a program or subroutine.	5.8
Compute	C:	Assigns a value to a variable.	5.9
Dimension	D:	Defines a numeric array or string variable.	5.10
Execute Indirect	XI:	Performs an instruction contained in a string.	5.11

Name	Operation Code	Meaning	Section
File Output	FO:	Writes a record to a diskette.	5.12
File Input	FI:	Reads a record from a diskette.	5.13
Graphics	G:	Displays non-text images to a student.	5.14
Sound	S:	Sends sound to a student.	5.15
Speech	V:	Sends speech to a student.	5.16
Workspace	W:	Gives current status of the numeric and character buffers.	5.17

The Graphics, Sound, and Speech instructions of TI PILOT are especially significant enhancements to the PILOT language. These instructions offer you the opportunity to use the TI Home Computer's advanced graphics, sound, and speech features to increase the effectiveness of a program.

NOTE: In Section 5 the discussion of each TI PILOT instruction begins with a line that shows its general format. Certain notational conventions have been used in the Format lines and the text refering to the format.

[ ] Brackets indicate optional items. You may use the items if you wish, but they are not required.

. . . An ellipsis indicates that the preceding item may be repeated as many times as you desire.

italics Italicized words indicate the kind of item or items to be used. Enter your own choice in place of the italicized words when you enter the instruction.

5.1 Problem--PR:

Format

PR: *option-list*

The Problem instruction marks the beginning of a program segment or the beginning of a new problem group. The PR: instruction has no modifiers, conditioners, or relational conditioners. However, it can have one or more option codes.

5.1.1 Options

The following option codes can be used with the Problem instruction:

Option Code	Action
U	Internally converts all the alphabetic characters entered by the student to upper-case characters. Answers appear on the screen exactly as typed. (If the U option is not used, the characters consist of upper- and lower-case characters as entered.)
L	Internally converts all the alphabetic characters entered by the student to lower-case characters. Answers appear on the screen exactly as typed. (If the L option is not used, the characters consist of upper- and lower-case characters as entered.)
S	Internally removes all spaces from the student's reply. This sometimes makes it easier to look for an expected phrase in a student's answer. Answers appear on the screen exactly as typed. (If the S option is not used, multiple spaces in the student's reply are automatically compressed into one space. Leading and trailing spaces are always removed.)
G	Allows the student to use the GOTO command during the program. (See section 6 for a description of the GOTO command.)
E	Allows the student to use an Escape command in the program. (See section 6 for a description of the Escape command.)
W	Discards the current label table. This lets you use duplicate labels. You can use the same label names once more after this point or until another PR: instruction with a W option is encountered.
B	Discards the current variables, both string and numeric. This allows you to reuse the variable space.

When a PR: instruction which has no options is performed, the options specified by the most previous PR: instruction are still in effect. If you want to change options, write a new PR: instruction to change one or more of the options. If an option is changed, however, then all other options must be specified again for them to remain in effect.

Options may be specified in any order with or without spaces. If both the U and L options are specified (in either order), all the alphabetic characters entered by the student are converted to upper-case.

5.1.2 Examples

The instruction

PR:

defines the beginning of a new program segment. No options are specified. If a previous PR: instruction had been issued with options, those options would remain in effect.

The instruction

PR: U

causes all the alphabetic characters entered by the student to be converted to upper-case characters.

The instruction

PR: LGS

causes all the alphabetic characters entered by the student to be converted to lower-case characters (the L option). Furthermore, the student is allowed to use the GOTO command during the program (the G command) and all spaces are removed from the student's reply (the S option).

5.2 Remark--R:

Format

R: any-text-or-spaces

or (any-instruction): text, destination, or expression : any-text-or-spaces

The Remark instruction lets you include comments within a program. The comments are not seen by the student. You can use it to document the program by explaining the logic of the program to someone who may be studying it. You can also use the Remark instruction to separate segments of your program with blank lines to improve its readability.

With TI PILOT there is a "trailing remark" capability (see alternate format above). If a colon is encountered a second time in a line, the text after it is considered a remark. Trailing remarks can be used with most TI PILOT instructions.

However, not all of the instructions in TI PILOT can have a trailing remark used with them. Trailing remarks cannot be used with the Accept instruction. If a trailing remark is used with a Type instruction, it will be displayed because all characters in the text-field are displayed. If a trailing remark is used with a Graphics or Sound instruction, the command-list must not end with a semicolon.

Remark instructions are ignored by the computer and, therefore, have no impact on the operation of a program. However, remarks do occupy storage space on a diskette.

Examples

The instruction

```
R: CALCULATE THE PERCENTAGE OF CORRECT ANSWERS
```

is a comment on the purpose of a program segment.

As another example, the R instruction can be used to improve the readability of a program.

```
R:
R:   CALCULATE THE PERCENTAGE OF CORRECT ANSWERS
R:
     C: P = (C1/T1) * 100
R:
```

A trailing remark is illustrated in the example below.

```
C: A = 5        : Value is assigned to a numeric variable.
```

5.3 Type--T:

Format

T: text-field

The Type instruction causes the text-field to be displayed on the display screen. If a text-field is not specified, a blank line is printed.

There are a number of types of data that can be specified in the text-field.

5.3.1 Text-Field Types

The text-field consists of numeric constants, numeric variables, string constants, string variables, and spaces.

5.3.1.1 Numeric Constants

When a numeric constant appears in the text-field, it is displayed on the screen.

For example, the instruction

```
T: 5.8
```

causes the numeric constant, 5.8, to appear on the display.

5.3.1.2 Numeric Variables

When a numeric variable appears in the text-field, its value is displayed. To be displayed, the numeric variable must be preceded by a number sign (#) and must be followed by at least one space. The numeric variable also must have been previously defined in a Compute or Dimension instruction. (See section 5.9 for a description of the Compute instruction and section 5.10 for a description of the Dimension instruction.) If a numeric variable in a text-field has not been previously defined or if the variable is not followed by at least one space, the variable's value is not displayed.

For example, the Compute instruction

```
*  C: Z = 1234.5
```

assigns the value 1234.5 to the numeric variable Z. Following this assignment, the instruction

```
    T: #Z
```

causes 1234.5 to be displayed.

If Z had not been defined by a Compute instruction or a Dimension instruction, then the instruction

```
    T: #Z
```

would cause the characters #Z to be displayed.

A numeric variable must be followed by at least one space, which is not displayed. If at least one space does not follow the variable, the value of the variable is not displayed. If the numeric variable is the last element in the text-field, however, it is not necessary to follow it with a space.

The numeric value in the text-field cannot be an array element unless you first assign the array element value to a numeric variable. For example, the instructions

```
    C: X = R(3)

    T: #X
```

assign the value in element 3 of array R to the numeric variable X and display the value.

5.3.1.3 String Constants

When a string constant appears in the text-field, it is displayed. For example, the instruction

```
T: THE ANSWER IS "WASHINGTON."
```

causes the following display:

```
THE ANSWER IS "WASHINGTON."
```

5.3.1.4 String Variables

To display its contents, a string variable must be preceded in the text-field by a dollar sign ($) and must be followed by at least one space. The variable also must have been previously defined by a Compute or Dimension instruction. (See section 5.9 for a description of the Compute instruction and section 5.10 for a description of the Dimension instruction.) If a string variable in the text-field has not been previously defined or if the variable is not followed by at least one space, the variable's value is not displayed.

For example, after first reserving space with a D: instruction

```
D: M$(6)
```

the instruction

```
C: M$ = "BEAGLE"
```

assigns the characters BEAGLE to the string variable M$.

Then, the instruction

```
T: ONE KIND OF LONG-EARED DOG IS A $M$ .
```

causes the following to be displayed:

```
ONE KIND OF LONG-EARED DOG IS A BEAGLE.
```

Notice that in the previous example the space following the string variable was not displayed. If you want a space to be displayed, put two spaces following the string variable.

The instruction

```
T: ONE KIND OF LONG-EARED DOG IS A $M$.
```

causes the following to be displayed:

```
ONE KIND OF LONG-EARED DOG IS A $M$.
```

because there is not a space following the string variable.

If M$ has not been previously defined, the instruction

```
T: A $M$ IS CLASSIFIED AS A HOUND.
```

would cause the following to be displayed.

```
A $M$ IS CLASSIFIED AS A HOUND.
```

5.3.2 Modifiers

The H modifier can be used with a Type instruction to suppress the carriage return and line feed which usually follow automatically after the text-field is displayed.

For example, the two instructions

```
T: HOW NOW,
T: BROWN COW?
```

cause the following two lines of text to be displayed.

```
HOW NOW,
BROWN COW?
```

But the instructions

```
TH: HOW NOW,              (a space follows the comma)
T: BROWN COW?
```

cause the following line to be displayed.

```
HOW NOW, BROWN COW?
```

5.3.3 Conditioners

Any of the conditioners can be specified with the Type instruction. (See section 3 for a discussion of the instruction conditioners.)

For example, the instruction,

```
TN: THAT'S NOT QUITE RIGHT.
```

causes the text-field to be displayed if the match condition is not true. (See section 5.5 for a discussion of the Match instruction.)

5.3.4 Relational Conditioners

Relational conditioners can also be specified for a Type instruction.

For example, the instruction

```
T(X=8): NOW, LET'S TRY SOMETHING DIFFERENT.
```

causes the text-field to be displayed if the numeric variable X is equal to 8. Otherwise, the text-field is not displayed.

As another example, the two instructions

```
T(X=8): NOW LET'S TRY SOMETHING DIFFERENT.
TC: A BILLY GOAT EATS GRASS.
```

cause the text-fields to be displayed only if the numeric variable X is equal to 8. Since the C conditioner refers to the last relational conditioner, the text, A BILLY GOAT EATS GRASS is displayed only if X is equal to 8.

5.3.5 Continuation of the Type Instruction

If you want to display several lines of text, you can use several Type instructions together. You need only include the T: instruction code in the first Type instruction. Subsequent type instructions require only the colon before the text-field. With continuation lines such as these, any modifiers, conditioners, or relational conditions set in the initial T: instruction apply as well to the continuation lines.

For example,

```
T(X=8): NOW LET'S TRY SOMETHING DIFFERENT.
: A BILLY GOAT EATS GRASS.
```

causes the text

```
NOW LET'S TRY SOMETHING DIFFERENT.
A BILLY GOAT EATS GRASS.
```

to be displayed if X equals 8. The conditional expression applies to both the first Type instruction and the continuation line.

5.3.6 Using the Type Instruction for Screen and Cursor Control

The Type instruction is normally used to cause lines of text to be displayed, but a limited form of screen and cursor control can be implemented on a video display.

These operations and their character codes are:

Operation	Code
Backward Space	8
Forward Space	137
Down a Line	10
Up a Line	139
Clear Screen and Home Cursor	12
Carriage Return	13

You can cause these operations to be performed by sending to the display screen the characters which cause these operations. You can assign these characters to a string variable and then use a Type instruction to send the characters to the display.

For example, the character code for a carriage return is 13 and the character code for a forward space is 9.

The following Compute instructions

```
C: F$ = CHR(9)
C: C$ = CHR(13)
```

assign the forward space character code to the string variable F$ and the carriage return character code to the string variable C$.

Then, the Type instruction

```
TH: A ------ IS A LONG-EARED HOUND.  $C$ $F$ $F$
```

prints the text A ------ IS A LONG-EARED HOUND. Then, it causes a carriage return back to the beginning of the line of text (the letter A) and forward spaces to the first hyphen so the student can fill in the missing word. The H modifier suppresses the automatic carriage return and line feed at the end of the line.

5.3.7 Examples

```
D: S$(1)
D: B$(1)
C: S$=CHR(32)
C: B$=CHR(8)
```

The following instructions

```
TH: THAT IS $S$ $B$
TH(C/8 > 2): NOT $S$ $B$
T: THE CORRECT ANSWER.
```

cause the text line THAT IS THE CORRECT ANSWER to be displayed unless the numeric value C divided by 8 is greater than 2. In that case, the text line THAT IS NOT THE CORRECT ANSWER is displayed.

As another example, assume the string variable N$ is equal to the characters BOB and that the numeric variable X is equal to 84.

The instruction

```
T: OK, $N$ , ASSUME THAT THE TEMPERATURE IS #X .
```

causes OK, BOB, ASSUME THAT THE TEMPERATURE IS 84. to be displayed.

5.4 Accept--A:

Format

```
A:
A: $string-variable$
A: #numeric-variable
```

The Accept instruction accepts a response from a student. The student's response is saved in the Answer Buffer (%B) and can be used for subsequent instructions.

The response can also be assigned to a string-variable or numeric-variable in the Accept instruction.

The student's response is edited by TI PILOT as follows.

!o! All leading and trailing spaces are removed.

!o! All multiple spaces within the response are compressed to a single space.

!o! If the S option was specified by the last PR: instruction (see section 5.1), all spaces are removed from the student's response.

!o! If the U option was specified by the last PR: instruction, all letters are converted to upper case. The U option has no effect on non-alphabetic characters.

!o! If the L option was specified by the last PR: instruction, all letters are converted to lower case. The L option has no effect on non-alphabetic characters.

5.4.1 Overriding Automatic Editing

You may want to accept a student's response in the exact form in which it was entered. You can do so with the X (eXact) modifier. The X modifier suppresses the automatic editing of the student's response and overrides any options specified by the last PR: instruction.

You can specify the eXact modifier with an Accept instruction like this:

```
AX:
```

5.4.2 System Variables (%B and %A)

Two system variables are automatically updated with each Accept instruction. One of these is the Answer Buffer (%B), and the other is an Answer Counter (%A).

The student's response is automatically placed in the internal Answer Buffer which you can reference by the system variable name %B. The buffer's contents are replaced with each new response and always contains the last student response. Therefore, if you want to save a student's response for later use, you must use a string variable and/or a numeric variable with the Accept instruction.

The Answer Counter lets you count the number of student replies. It counts the number of times that the same Accept instruction is performed without an intervening Accept instruction. The first time that a particular Accept instruction is performed, the Answer Counter is set to one. If the same Accept instruction is performed without an intervening Accept instruction, the Answer Counter is incremented by one each time it is performed. Then, when a different Accept instruction is performed, the Answer Counter is reset to one.

You can access the number in the Answer Counter and determine how many times a student has responded to a question. You can use the value of %A in an expression or you can set it to a specific value. For example, consider this program segment.

```
D: R$(10)
C: N=0
T: WHAT IS THE NAME OF THE COUNTRY IMMEDIATELY
 : NORTH OF THE UNITED STATES?
*REPLY
A: $R$
C: N = %A
M: CANADA
TN: NO, $R$  IS NOT CORRECT.
TN1: THIS IS YOUR #N ST. INCORRECT ANSWER.
TN2: THIS IS YOUR #N ND. INCORRECT ANSWER.
TN3: THIS IS YOUR #N RD. INCORRECT ANSWER.
TN(%A>3): THIS IS YOUR #N TH. INCORRECT ANSWER.
JN: REPLY
TY: GOOD.  $R$  IS NORTH OF THE UNITED STATES.
```

The program poses a question to the student and analyzes the student's answer.

The instruction, TN: NO, R IS NOT CORRECT. causes a message to be displayed if the student's answer is wrong. (Note that an extra space is necessary after R for proper spacing of the displayed message.) If the student enters MEXICO as an answer, the following message is displayed.

```
NO, MEXICO IS NOT CORRECT.
```

Furthermore, the instructions which use the %A system variable in a relational expression cause a number to be displayed. The instruction, TN(3): THIS IS YOUR #N RD. INCORRECT ANSWER. causes a message to be displayed if it is the student's third wrong answer (the same Accept instruction has been performed 3 times). The message displayed is

```
THIS IS YOUR 3RD. INCORRECT ANSWER.
```

NOTE: The largest value the Answer Counter can hold is 9.

5.4.3 Using Variables to Accept a Student's Response.

You can put a string-variable and/or a numeric-variable in the text-field of an Accept instruction. The student's response is stored both in the variable you specify and in the Answer Buffer (%B).

When a string variable is placed in the text-field, the student's response is assigned to the string variable. A string variable must be preceded by a dollar sign ($), and the string variable must have been previously defined in a Compute or Dimension instruction (see section 5.9 for a description of the Compute instruction and section 5.10 for a description of the Dimension instruction).

As an example, the instruction

```
A: $R$
```

causes the student's response to be placed in the string variable R$. R$ must have been previously defined with a Dimension instruction (see section 5.10 for a description of the Dimension instruction).

When a numeric variable is specified in the text-field of an Accept instruction, the first decimal value in the student input is placed in the numeric variable. The numeric variable must have been preceded by a number sign (#) and must be previously defined in a Compute instruction or a Dimension instruction.

For example, the instruction

```
A: #I
```

causes the first decimal value in the student response to be placed in the previously defined numeric variable I. If the student's reply is "It is 18 feet and 2 inches," the value 18 is placed in I.

If a numeric variable is specified and the student's response does not include a number, the Error condition is set. Subsequent instructions can then check the Error condition to determine the appropriate action.

For example, if the student's reply is "It is eighteen feet and two inches," then the Error condition is set. Here is an example of a series of instructions.

```
*ASK    T: WHAT IS 2 + 5?
A: #I
TE: PLEASE TYPE A NUMBER
JE: ASK
```

The first instruction (labeled ASK) displays the question WHAT IS 2 + 5?

The second instruction accepts the student's answer and places the number in the answer into the numeric variable I.

If the student does not enter a number (for example, if the student enters SEVEN or simply pressed <return>), the third instruction displays the message, PLEASE TYPE A NUMBER.

The fourth instruction causes a jump to the first instruction (labeled ASK) if the student does not enter a number.

The third and fourth instructions both use the Error conditioner (E). The Error condition must be set to cause these instructions to be performed.

If the student enters a number, the third instruction does not display a message and the fourth instruction does not cause a jump.

Numeric values in a student response can be expressed in a variety of ways, all of which are correct. For example, the value of pi could be expressed as 3.14, 3.1416, or 3.14159..., depending upon the degree of precision required.

You can permit the student some flexibility in the answer by editing his response.

For example, the instructions

```
T: WHAT IS THE VALUE OF PI?
A: #I
TH(ABS(I-3.14) > 0.01): TRY AGAIN AND BE A LITTLE MORE PRECISE.
```

would print a message if the student's answer is not within 0.01 of the correct value.

You can specify both a string variable and a numeric variable in the text-field of an Accept instruction. The two can be specified in any order, but they must be separated by at least one space. If both are specified, all of the edited student response is placed in the string variable and the first decimal number in the response is placed in the numeric variable.

5.4.4 Accept Single

You can use the Accept Single modifier (S) to accept only a single character from a student. With the S modifier, as soon as the student presses a key on the keyboard, the character is stored in the %B Answer Buffer (and any numeric variable or string variable specified) and the next instruction is performed. The cursor remains where it is after the character is entered.

For example, the instructions

```
T: STUDY THE CHART ON PAGE 68 OF YOUR WORKBOOK
 : AND THEN PRESS ANY KEY TO CONTINUE.
AS:
```

give directions to the student and then wait for the student to press a key before proceeding. No automatic editing occurs on the response to an AS:.

5.4.5 Examples

The following program segment demonstrates the Accept instruction and the Answer Counter.

```
D: R$(15)
T: WHO WAS THE FIRST PRESIDENT OF THE UNITED STATES?
*REPLY  A: $R$
M: WASHINGTON
TY: GOOD!
JY: CONT
T1: NO, $R$  IS NOT QUITE RIGHT.  TRY AGAIN
T2: NO--OUR NATION'S CAPITAL IS NAMED AFTER THIS MAN.
T3: HIS NAME IS WASHINGTON.
J(%A < 3): REPLY
*CONT
```

The numbers following the Type instructions (T:) are conditioners which refer to the value in the Answer Counter (%A). In other words, T1: is equivalent to T(%A=1): in this program segment.

The first instruction asks a question. The second instruction (labeled REPLY) accepts the student's answer and automatically saves the answer in the system variable %B. The third instruction matches the student's reply with the correct answer (see section 5.5 for a description of the Match instruction). If the student's answer is correct, the fourth instruction causes the message, GOOD! to be displayed and the fifth instruction causes a jump to the label CONT.

If the student's first answer is not correct, the sixth instruction is performed. It informs the student that the response was incorrect. If this is the student's first wrong answer, neither the seventh, eighth, or ninth instruction is performed. The tenth instruction causes a jump back to the Accept instruction and awaits another answer from the student.

If the student's second answer is not correct, the fourth and fifth instructions are not performed. The sixth instruction is not performed because the Answer Counter (%A) is now 2. The seventh instruction is performed. The eighth and ninth instructions are not performed. The tenth instruction causes a jump back to the Accept instruction.

If the student's third answer is not correct, the sixth and seventh instructions are not performed because the Answer Counter (%A) is now 3, not 2 or 1. In this case, the correct answer is displayed by the eighth instruction, and the ninth instruction causes a jump to the label CONT.

As another example, consider these instructions.

```
D: N$(15)
C: L=0
T: WHAT IS THE NAME OF YOUR BEST FRIEND?
AX: $N$
T: OK, SUPPOSE $N$  HAD 85 CENTS AND YOU BORROWED A
 : QUARTER.  HOW MUCH WOULD $N$  HAVE LEFT?
*REPLY   A: #L
TE: ENTER A NUMBER, PLEASE.
JE: REPLY
```

The first instruction is a prompt for the student to enter a name. The second instruction puts the name entered by the student into the string variable N$. The third and fourth instructions (a Type instruction and a Type continuation) pose a question and cause the name entered by the student to be displayed as part of the question. The fifth instruction (labeled REPLY) accepts the student's answer. If the student does not include a number in the reply, the sixth instruction displays the message ENTER A NUMBER, PLEASE. and the seventh instruction causes a jump to the instruction labeled REPLY. If the student enters a number, nothing is displayed (sixth instruction) and no jump occurs (seventh instruction).

5.5 Match—M:

Format

M: text-field

The Match instruction lets you analyze a student's answer. The text-field is compared to the student's answer stored in the Answer Buffer %B. When a match is found, the Y conditioner is set. When a match is not found, the N conditioner is set. The state of the conditioner can then be used in subsequent instructions.

As an example, examine the following instructions.

```
T: A "PAIR" IS TWO OF SOMETHING.
T: NAME A PAIR OF THINGS ON YOUR HEAD.
A:
M: EARS
*OK   TY: GOOD.
JY: CONT
M: EYES
JY: OK
T: NO, HOW ABOUT EARS OR EYES?  YOU HAVE TWO OF THOSE.
*CONT
```

The first two instructions pose the question. The third instruction accepts the student's answer. The fourth instruction compares the student's answer to one of the two expected answers, "EARS." If there is a match, the Y conditioner is set and the fifth and sixth instructions cause "GOOD." to be displayed and program control to jump to the label CONT. If the student's answer is not "EARS," then the seventh instruction compares the student's answer with the second expected answer, "EYES." If there is a match, the Y conditioner is set and the eighth instruction causes a jump to the fifth instruction. If there is not a match on the second Match instruction, the ninth instruction displays a message and program control passes to the label CONT.

Remember to structure the text-field of a Match instruction in accordance with any editing of the input by the Accept instruction. Recall that the PR: instruction can specify the U, L, or S options. One of the two space-removing options (compress multiple spaces to a single space or remove all spaces) is always in effect unless the Accept instruction specifies an exact match (AX:).

5.5.1 Match Instruction Special Features

There are several special features which can be specified in the text-field of a Match instruction which give the student more or less flexibility in satisfying a required response.

The following special features are available with the Match instruction.

SYMBOL	DEFINITION
&	Joins two or more required words in an answer; disregards any characters which occur between the required sequence of characters.
!	Separates the choices when there is more than one acceptable answer.
%	Used before and/or after a required word; causes a match only if there is a space within the response or at the beginning or end of the response where the % appears.
*	Matches any single character which appears at that same relative position in the student's answer.

5.5.1.1— The & Matching Feature

An ampersand (&) in the text-field of a Match instruction allows you to specify a match with more than one word or character string in the student's answer. For a correct match to occur, each word or character string joined together by the ampersand in the Match instruction must be present and in the same order in the student's response.

The & disregards any characters that might be present between the words. For example, the instruction

M: EARS&EYES

causes the Y conditioner to be set for any answer with the word EARS followed by the word EYES. All of the following student responses cause the Y conditioner to be set.

EARS AND EYES
A PAIR OF EARS AND EYES
BEARS HAVE EYES

NOTE: The Match instruction ignores the characters in the student response until it finds the first occurrence of the word EARS; therefore, the response BEARS HAVE EYES causes the Y conditioner to be set.

The following responses do not cause a match because the words are not in the same order.

EYES AND EARS
POTATOES HAVE EYES, BUT NO EARS

With the & matching feature, you can specify more than two key words.

For example,

M: THUNDER&LIGHTNING&RAIN

causes a match for a student response of

WITH THUNDERING RAGE AND LIGHTNING REFLEXES, HE RAINED BLOW AFTER BLOW ON ME.

5.5.1.2 The ! Matching Feature

The ! feature lets you check for one of several expected words in an answer. The expected words must be separated by an exclamation mark (!) in the text-field of the Match instruction. If any one of the words appears in the student's answer, the Y conditioner is set.

For example,

```
T: NAME A PRESIDENT WHOSE LIKENESS APPEARS ON THE
: MOUNT RUSHMORE NATIONAL MONUMENT.
A:
M: LINCOLN!JEFFERSON!ROOSEVELT!WASHINGTON
```

causes the Y conditioner to be set for any student answer including LINCOLN, JEFFERSON, ROOSEVELT, or WASHINGTON.

Note: It is important that you do not put spaces on either side of the exclamation mark, since this will cause the exclamation mark to be included as part of the required response. It is also important that you do not place an exclamation mark after the last word, since this will cause any response to match.

5.5.1.3 The % Matching Feature

When the % character is used immediately preceding a word in the text-field of a Match instruction, the word(s) in the student response must have a space in that position or must be the first characters typed in the student response.

If the % character immediately follows a word in the text-field of a Match instruction, the student response must have a space in that position or must be the last characters typed in the student response.

For example,

```
M: %PEN
```

causes a successful match for the following student responses.

```
PEN
PAPER AND PENCIL
PENTHOUSE
```

but would not cause a match for the following responses.

```
HAPPEN
SPENT
TURPENTINE
```

As another example,

```
M: PEN%
```

would cause a match for the following inputs:

```
MISSHAPEN
PEN PAL
THE DOOR WAS OPEN
```

but does not cause a match for these responses.

```
SPENT
PENNY
REPENTANT
```

When % is specified both immediately before and after a word, the word must appear exactly as specified.

For example,

```
M: %PEN%
```

causes a match for the following student responses.

```
A PEN AND PENCIL CASE
HE LEFT THE PEN GATE OPEN
```

but does not cause a match for these responses.

```
IT MAY HAPPEN
SHE SPENT IT ALL
```

5.5.1.4 Combinations of &, !, and %

You can use these special editing features to screen the student answers. For example,

```
M: PLANE%&%UP%&SKY!AIR
```

causes a match with the following

```
A GLIDER IS HEAVIER THAN AIR
AN AIRPLANE GOES UP IN THE BLUE SKY
A PLANE BLEW UP IN THE SKY
A PLANE BLEW UP IN THE AIR
```

but not a match with the following

```
PLANES FLY UP IN THE SKY
A PLANE LIFTED UPWARD INTO THE SKY
THE PLANE FLEW UP INTO THE SUN
```

As another example,

```
M: CARGO&PLANE%
```

causes a match with the following,

```
THE CARGO AIRPLANE WAS TOO HEAVY TO LIFT OFF.
```

but not a match with the following,

```
THE AIRPLANE WAS TOO BIG.
```

or with,

```
THE CARGO AIRPLANES FLEW TOGETHER.
```

5.5.1.5 The * Option

When an asterisk appears within a word in the text-field of a Match instruction, it is matched with any single character entered by the student in the same relative position within the answer. This technique allows the student some flexibility for simple spelling or typing errors. For example,

```
T: WHAT IS THE STATE CAPITAL OF TEXAS?
A:
M: AUST*N
```

will match with the following inputs

```
AUSTIN
AUSTEN
AUSTYN
I THINK IT IS SAN AUSTEN BUT IT MAY BE DALLAS
```

5.5.2 The S Modifier

The asterisk feature provides for expected spelling errors in a single character position. However, you can allow more general spelling errors with the S modifier. The S modifier is placed after the Match instruction code (MS:). The S modifier allows a correct match for common types of spelling errors, such as one wrong character in a word or two characters inverted in a word.

For example,

```
T: WHAT IS AN ELECTRONIC MACHINE THAT DOES FAST CALCULATIONS?
A:
MS: COMPUTER
```

causes a match with the following responses.

```
COMPUTER
COMPUTIR
COMPUTRE
```

5.5.3 Automatic Jump Option

You can use the automatic jump option with the Match instruction to simplify the coding of a program.

When the jump option (J) is specified with the M: operation code, it causes an automatic jump to the next Match instruction if the match fails.

For example, the following program uses the automatic jump feature.

```
T: WHAT DID CAPTAIN AHAB THROW AT MOBY DICK?
A:
MJ: HARPOON
T: RIGHT.
J: NEXT
MJ: LANCE
T: YOU MIGHT CALL IT THAT.  IT'S A "HARPOON."
J: NEXT
M: SPEAR
TY: IT'S A KIND OF SPEAR CALLED A "HARPOON."
JY: NEXT
TN: IT'S CALLED A "HARPOON."
*NEXT
```

The first instruction poses a question to the student and the second instruction accepts the student's response. The third instruction matches the response with the correct answer, HARPOON. If the response is correct, the fourth instruction displays the message RIGHT. and the fifth instruction causes a jump to the label NEXT. If the response is not correct, the third instruction causes an automatic jump to the next Match instruction (the sixth instruction).

The sixth instruction matches the response with LANCE. If the response is LANCE, the seventh instruction displays YOU MIGHT CALL IT THAT. IT'S A "HARPOON." and the eighth instruction causes a jump to the label NEXT. If the response is not LANCE, the sixth instruction causes an automatic jump to the next Match instruction (the ninth instruction).

The ninth instruction matches the response with SPEAR. If the response is SPEAR, the tenth instruction displays IT'S A KIND OF SPEAR CALLED A "HARPOON." and the eleventh instruction causes a jump to the label NEXT. If the response is not SPEAR, the twelfth instruction displays IT'S CALLED A "HARPOON".

5.5.4 Examples

The following program segment illustrates the use of several Match instruction special features.

```
PR: U
T: WHO WROTE "THE STAR SPANGLED BANNER"?
*REPLY A:
M: FRANC*S%&S&%KEY%
TY: THAT'S CORRECT.
JY: CONT
TN: NOT QUITE.  TRY AGAIN.
JN: REPLY
*CONT
```

The first instruction poses the question. The second instruction (labeled REPLY) accepts the student's reply. The third instruction matches the student's reply against the key words. The answer must contain three words and in the correct order. The first key word is FRANC*S. Notice that the asterisk allows for a possible spelling error; for example, FRANCES instead of FRANCIS. The second key word could be SCOTT, the expected answer, or simply the initial, S. The third key word must be spelled exactly right.

If the answer is correct, the fourth instruction displays the message, THAT'S CORRECT, and the fifth instruction causes a jump to the instruction labeled CONT.

If the answer is not correct, the sixth instruction displays the message NOT QUITE. TRY AGAIN. Then the seventh instruction causes a jump to the Accept instruction and the student is once again prompted for a reply.

Here is a program segment with the automatic jump feature.

```
PR: U
*COLORS
T: NAME ONE OF THE COLORS IN THE UNITED STATES FLAG.
A:
MJ: RED
T: GOOD.  WHAT PART OF THE FLAG IS RED?
J: REDA
MJ: WHITE
T: YES, WHITE IS ONE COLOR.  WHAT DOES WHITE REPRESENT?
J: WHITEA
M: BLUE
TY: THAT'S RIGHT.  WHAT PART OF THE FLAG IS BLUE?
JY: BLUEA
T: THE FLAG HAS THREE COLORS: RED, WHITE, AND BLUE.
J: COLORS
*REDA
     .
     .
     .
*WHITEA
     .
     .
     .
*BLUEA
```

In this example, if there is not a correct match for RED or WHITE, the MJ: instruction causes an automatic jump to the next Match instruction.

5.6 Jump--J:

Format

```
J: destination
```

The Jump instruction alters the normal sequential performance of instructions and causes the computer to transfer program control to the instruction specified by the destination in the text-field. A Jump instruction may be conditional or unconditional. When used in a program, conditional Jump instructions allow you to make decisions of what to do next.

5.6.1 Destinations of a Jump Instruction

The destination in the text-field specifies the instruction to which a jump is made. The destination must be one of the four items below.

!o! a label

!o! @A

!o! @M

!o! @P

5.6.1.1 A Label

The destination can be a label associated with a specific instruction. For example, the instruction

```
J: CONT
```

causes a jump to the instruction labeled CONT.

5.6.1.2 @A

If the destination-field is @A, a jump is made to the last Accept instruction which was performed.

For example, the jump instruction in the following program segment causes a jump to the last Accept instruction when the answer is incorrect.

```
T: WHAT WAS THE NAME OF COLUMBUS'S FLAGSHIP?
*ANSWER
A:
M: SANTA MARIA
TN: NO, TRY AGAIN.
JN: @A
```

A JN: ANSWER instruction would accomplish the same thing. Using a label in the destination, however, requires more memory space than @A (or @M or @P).

5.6.1.3 @M

If the destination is @M, a jump is made to the next Match instruction.

For example, in the following program segment, the instruction JN: @M causes a jump to the next Match instruction when the answer is incorrect.

```
*SHIPS
T: WHAT WAS THE NAME OF COLUMBUS'S FLAGSHIP?
*ANSWER
A:
M: SANTA MARIA
JN: @M
T: GOOD.
J: CONT
M: NINA!PINTA
TY: THAT WAS ONE OF HIS SHIPS, BUT NOT THE FLAGSHIP.
JY: SHIPS
*CONT
PR:
```

5.6.1.4 @P

If the destination is @P, a jump is made to the next Problem instruction. In the previous example, the instruction J: CONT could just as well have been J: @P.

5.6.2 Using Conditioners with the Jump Instruction

A Jump instruction can be either conditional or unconditional. An unconditional Jump instruction always causes a transfer of program control when it is performed. A conditional instruction, on the other hand, may or may not cause a transfer of program control, depending upon the state of the conditioner specified by the Jump instruction.

The following conditioners can be used for conditional Jump instructions.

- The Yes conditioner (Y)
- The No conditioner (N)
- The Error conditioner (E)
- The Last Relational conditioner (C)
- The Digit conditioner (a number)

5.6.2.1 The Yes (Y) and No (N) Conditioners

The Y conditioner causes a jump if the Y conditioner was set as a result of a previous Match instruction. The N conditioner causes a jump if the N conditioner was set as a result of a previous Match instruction.

For example, in the following program segment

```
T: WHAT WAS THE NAME OF COLUMBUS'S FLAGSHIP?
A:
M: SANTA MARIA
JY: RIGHT
JN: WRONG
```

the JY: instruction causes a jump to the instruction labeled RIGHT if the student enters SANTA MARIA and the JN: instruction causes a jump to the instruction labeled WRONG if the student enters a response other than SANTA MARIA. In this example, it is not necessary to use the N conditioner with the JN: instruction since the Jump instruction is performed anyway if the Y condition is not set.

5.6.2.2 The Error (E) Conditioner

The Error conditioner causes a transfer of program control if the Error condition was set as the result of a previous instruction operation.

The Error conditioner is set when an Accept instruction specifies a numeric variable and the student does not enter a number.

5.6.2.3 The Last Relational Conditioner (C)

The Last Relational Conditioner causes a jump if the Last Relational Condition was set as the result of a previous instruction operation.

For example, in the following program segment

```
T(N < 4): TRY A LARGER NUMBER
JC: TOOSML
```

if the numeric variable N is less than 4, the Relational Conditioner is set and the Type instruction is performed. The C conditioner attached to the Jump instruction causes the relational conditioner to be checked by TI PILOT. Since it is set, the Jump instruction causes a transfer of program control to the label TOOSML.

5.6.2.4 The Digit Conditioner

The digit conditioner causes a jump if the value of the digit attached to the Jump instruction is equal to the value in the Answer Counter (%A). The Digit conditioner must be a number from 1 to 9.

The Answer Counter is set to 1 whenever a specific Accept instruction is performed the first time. The Answer Counter is incremented by 1 each time that the same Accept instruction is performed without an intervening Accept instruction.

As an example, in the program segment

```
C: N=0
T: WHAT IS THE NAME OF THE PLANET NEAREST TO OUR SUN?
 : YOU MAY HAVE THREE GUESSES, AND THEN STRIKE THREE,
 : YOU'RE OUT.
A:
M: MERCURY
TY1: GOOD.  YOU GOT IT THE FIRST TIME.
TY2: OK.  YOU GOT IT THE SECOND TIME.
TY3: NOT BAD.  YOU GOT IT THE THIRD TIME.
JY: GOOD
J3: BAD
C: N=%A
T: NO, STRIKE #N .  TRY AGAIN.
J: @A
*BAD
T: NO, STRIKE 3.  YOU'RE OUT.  THE PLANET NEAREST
 : OUR SUN IS MERCURY.
```

In this example, the Digit conditioner with the Jump instructions (JY1:, JY2:, and J3:) causes a jump only if the value of the Digit conditioner is equal to the Answer Counter's value. Notice that the Digit conditioner also is used with the Type instruction. When an instruction has more than one conditioner, all conditions must be true in order for the instruction to be performed.

5.6.3 Relational Expressions

A Jump instruction can have a relational expression. A jump is performed if the relational expression is true.

For example,

```
J(X/12 < 5): LESS
```

causes a jump if the numeric variable X divided by 12 is less than 5. Otherwise, a jump is not made.

When an instruction has one or more conditioners and a relational expression, all the conditions and the expression must be true in order for the instruction to be performed.

As an example, the instruction

```
J3(X/12 < 5): LESS
```

causes a jump only if the Answer Counter is 3 and the numeric variable X divided by 12 is less than 5. Otherwise, a jump is not made.

The following program segment illustrates the use of expression evaluation:

```
C: N=0
T: YOU HAVE A BOX WITH 60 DOUGHNUTS IN IT.
T: HOW MANY DOZEN DOUGHNUTS DO YOU HAVE?
A: #N
JE: NONUM
M: 5
J(N < 5): LESS
J(N > 5): MORE
T: THAT'S RIGHT.
J: @P
*NONUM
T: ENTER A NUMBER
J: @A
*LESS
T: NO, YOU HAVE MORE THAN THAT.  TRY AGAIN.
J: @A
*MORE
T: NO, YOU DON'T HAVE THAT MANY.  TRY AGAIN.
J: @A
```

5.7 Use--U:

Format

U: destination-field

The Use instruction causes the computer to transfer program control to a subroutine. Like the Jump instruction, the Use instruction alters the normal sequential performance of instructions and causes program control to transfer to the instruction specified in the destination-field. The Use instruction causes one other event to happen, however. It causes the address (location) of the instruction following the Use instruction to be "remembered." The address of that instruction is saved on a subroutine stack.

The destination-field of a Use instruction can be any one of the four destinations which can be used by a Jump instruction: a label, the previous Accept instruction (@A), the next Match instruction (@M), or the next Problem instruction (@P).

When the subroutine named in the destination-field finishes, an End (E:) instruction at the end of the subroutine causes the address which was saved on the stack to be "popped" off the stack, and program control is returned to that address; that is, the address of the instruction following the U: instruction. It is also possible for the E: instruction to transfer program control to a different instruction. (See section 5.8 for a description of the End instruction.)

When you write a program which includes subroutines, make sure that program control does not transfer to a subroutine other than through the Use instruction.

If program control passes to a subroutine without its being called by a Use instruction, the End instruction at the end of the subroutine will pop an address value off of the stack even though an address value was not placed on the stack when the subroutine received control.

Examples

Consider the following program segment:

```
T: WHO WAS THE COMMANDER OF THE CONFEDERATE FORCES?
U: COMNDR
*COMNDR
A:
M: LEE
TN: NO, TRY AGAIN.
JN: @A
T: THAT'S RIGHT.
E:
```

In this program, The U: instruction calls the subroutine labeled *COMNDR. If the student's answer is wrong, the JN: instruction causes a jump to the last Accept instruction. If the student's answer is correct, the E: instruction returns program control to the instruction following the U: instruction.

When the E: instruction is first executed, the program branches to the label *COMNDR because that is the next address (instruction location) that is popped off the subroutine stack. When the E: instruction is executed the second time, the program ends and the computer returns to the p-System prompline. The second time through the E: instruction is understood to be the end of the program regardless of any instructions that may follow it.

The following program segment uses the U: instruction to calculate the number of months in a given number of years.

```
C: Y = 0
C: M = 0
C: G = 0
T: HOW MANY YEARS OLD ARE YOU?
A: #Y
TE: ENTER A NUMBER.
JE: @A
U: CALC
T: HOW MANY MONTHS ARE THERE IN #Y  YEARS?
A: #M
TE: ENTER A NUMBER.
JE: @A
T(M<>G): NO, TRY AGAIN.
JC: @A
T: THAT'S RIGHT.
J: @P
     .
     .
     .
*CALC
C: G = Y * 12
E:
     .
     .
     .
PR:
```

5.8 End--E:

Format

E: [destination-field]

The End instruction ends a program or a subroutine. If the End instruction is at the end of a program, it ends that program. If the End instruction is at the end of a subroutine, it ends that subroutine.

Just as a PR: instruction is normally used to mark the beginning of a program, the E: instruction is used to mark the end of a program. When it is used at the end of a program, the E: instruction causes a return to the p-System promptline.

The E: instruction is also used to mark the end of a subroutine within a program. The subroutine is called with the U: instruction (See section 5.7). When used at the end of a subroutine, the E: instruction causes the computer to exit from the subroutine and return to the instruction immediately following the U: instruction which called the subroutine.

TI PILOT maintains a subroutine stack. When a subroutine is called (with a U: instruction), the address of the instruction following the U: instruction is pushed onto this stack. Then, when an E: instruction is performed, it "pops" an address off the top of the stack (removes the most recent entry) and transfers program control to the instruction at that address. If there are no entries on the stack, the E: instruction terminates the program.

5.8.1 Destination-Field

The optional destination-field can be used to transfer control to another place in a program, rather than the instruction following a U: instruction. This option allows the program to remove entries from the subroutine stack without returning to the calling location. If an E: instruction with a destination-field is performed and there are no entries on the subroutine stack, the instruction ignores the destination-field and returns to the p-System.

The destination-field can contain an instruction label, an Accept instruction, a Match instruction, or a Problem instruction. These are the same four choices available with a Jump instruction. (See section 5.6 for a discussion of the Jump instruction and these choices.)

5.8.2 Examples

The following instruction

E:

returns the computer to the instruction following the most recently performed U: instruction or exits from the program if there are no entries on the subroutine stack.

The instruction

```
E: GIZARD
```

causes a jump to the label GIZARD and pops the most recently entered value from the subroutine stack.

5.9 Compute--C:

Format

C: variable = expression

The Compute instruction assigns a value to a variable. The variable on the left of the equals sign (=) is assigned the value of the expression on the right of the equals sign.

A Compute instruction may use either numeric or string data. (See section 6 for a description of the types of numeric and string data.)

5.9.1 Numeric Constants

A Compute instruction can include numeric constants. Numeric constants are used to assign a value to a numeric variable or numeric array element.

5.9.2 Numeric Variables

A numeric variable can be assigned a numeric value by the Compute instruction*. For example,

```
C: N = 8
```

assigns the value 8 to the numeric variable N. The instruction

```
C: N5 = 12.7
```

assigns the value 12.7 to the numeric variable N5. The instruction

```
C: X = 87E-2
```

assigns the value 0.87 to the numeric variable X. See section 7 for a discussion of floating point.

*NOTE: Numeric variables must be initialized with either a Compute or Dimension instruction. Otherwise, the variable name will not be recognized by the computer.

5.9.3 Numeric Arrays

A numeric array can be used in a Compute instruction. For example,

```
C: Z(3,8) = 5
```

assigns the value 5 to the element in the third row and eighth column of array Z. As another example,

```
C: N4 = R(5,2)
```

assigns the value of array element R(5,2) to the numeric variable N4.

Before a numeric array can be used in a Compute instruction, space must be reserved for the array by the Dimension instruction (see section 5.10).

5.9.4 Numeric Functions

A numeric function carries out a numerical calculation on a numeric value. The function is specified by a key word and the numeric value on which the calculation is performed is included in parentheses following the key word.

For example, the instruction

```
C: X = ABS(Z)
```

calculates the absolute value of the numeric variable Z and assigns that value to the numeric variable X.

Section 7 describes the numeric functions available with TI PILOT.

5.9.5 String Constants

A string constant can be included in a Compute instruction, which is used to assign a value to a string variable, or the string constant can be used in a string expression.

5.9.6 String Variables

A string variable can be assigned a value by a Compute instruction. For example,

```
C: Z$ = "VWXYZ"
```

assigns the string value VWXYZ to string variable Z$.

A string variable cannot be used in a Compute instruction until space has been allocated for it by the Dimension (D:) instruction (see section 5.10 for a description of the Dimension instruction). The D: instruction establishes the maximum length for the string variable. The length of the string variable can be from 1 through 256 characters.

The characters in a string are stored from left to right. Should the maximum length of the string variable not be large enough to contain the characters assigned to it, characters are truncated (deleted) on the right.

For example, the Dimension instruction

```
D: Z$(10)
```

assigns the string variable Z$ a maximum length of 10.

The instruction

```
C: Z$ = "PQRSTUVWXYZ"
```

causes the ten characters PQRSTUVWXY to be stored in Z$. The Z is truncated.

At this point, the current length of Z$ is equal to the maximum length (10).

The instruction

```
C: Z$ = "JKL"
```

causes the characters JKL to be stored in Z$. The current length of Z$ is 3.

The length of a string is changed whenever a different number of characters is stored in it.

5.9.7 String Pseudo-variables

A string pseudo-variable defines a subset of a string and is designated by a string variable name followed by a subscript. Only the part of the string variable specified by the subscript is changed; the rest of the string variable remains the same.

For example, assuming that space has already been reserved for Z$ with a Dimension instruction,

```
C: Z$ = "DOG & PIG"
C: Z$(7,3) = "CAT"
```

results in Z$ = "DOG & CAT"

For another example, the Dimension instruction

```
D: X$(15)
```

assigns a maximum length of 15 to the string variable X$.

Then the instruction

```
C: X$ = "HORSES AND COWS"
```

assigns HORSES AND COWS to X$. The instruction

```
C: X$(12,6) = "CATTLE"
```

results in the error message B-ERROR. This means that the subscript 6 is out of bounds. The C: instruction attempts to assign the characters L and E to positions 16 and 17, but space is reserved for only 15 positions by the D: instruction. To do this operation, the D: instruction must reserve space for at least 17 characters for the string variable X$.

The value assigned to a string variable may be padded on the right with spaces if necessary. For example, with X$ = HORSES AND COWS, the instruction

```
C: X$(1,6) = "PIGS"
```

results in X$ = PIGS AND COWS (three spaces between PIGS and AND).

The length of a string variable does not change as the result of using a pseudo-variable.

5.9.8 String Functions

A string function performs an operation on a string. The function is specified by a key word and the name of the string on which the operation is performed is enclosed in parentheses after the key word.

For example, the following instruction

```
C: X = LEN(R$)
```

calculates the current length of the string variable R$ and assigns that number to the numeric variable X.

Section 7 describes the string functions available with TI PILOT.

5.9.9 System Variable %A (Answer Counter)

A Compute instruction can include the system variable %A.

For example,

```
C: N = %A + 2
```

assigns the value 7 to N if the value in %A is currently 5.

5.9.10 System Variable %B (Answer Buffer)

A Compute instruction can include the system variable %B.

For example,

```
C: S$ = %B(10,3)
```

assigns to string variable S$ the tenth, eleventh, and twelfth characters in the Answer Buffer.

5.9.11 Expressions

An expression in a Compute instruction may contain any of four types of operators: arithmetic operators, relational operators, logical operators, and string operators.

5.9.11.1 Arithmetic Operators

An expression can contain the following arithmetic operators:

Symbol	Operation	Example
**	Exponentiation	X**Y (X to the Yth power)
*	Multiplication	X*Y
/	Division	X/Y
+	Addition	X+Y
-	Subtraction	X-Y

Plus (+) and minus (-) can be used to indicate positive and negative numbers. Numbers are assumed to be positive, unless indicated otherwise.

5.9.11.2 Relational Operators

Relational operators cause a comparison of two numbers or strings. The two items which are compared must be of the same kind (both numeric or both string). The result of a relational operation is either the numeric value 1 (indicating a true condition) or the numeric value 0 (indicating a false condition).

RELATIONAL OPERATORS

Symbol	Meaning	Example
=	equals	X=Y
<	less than	X<Y
>	greater than	X>Y
<>	not equal to	X<>Y
<=	less than or equal to	X<=Y
>=	greater than or equal to	X>=Y

5.9.11.3 Logical Operators

An expression can contain logical operators. Although logical operators can be used with any numeric value, they are most often used with the numeric results of relational operators. The result of a logical operation is either the numeric value 1 (indicating a true condition) or the numeric value 0 (indicating a false condition). When the logical operators evaluate a numeric value, any non-zero value is defined as a true condition.

LOGICAL OPERATORS

Symbol	Meaning	Example
tilde	Not X. If X is 0, then X is 1. If X is non-zero, then X is 0.	tilde X
&	X and Y. X&Y is 1 (true) if and only if X is non-zero and Y is non-zero. If either X or Y is 0, X&Y is 0 (false).	X&Y
!	X or Y. X!Y is 1 (true) if either X is non-zero or Y is non-zero, or both are non-zero. If both X and Y are equal to 0, X!Y is 0 (false).	X!Y

5.9.11.4 String Operator

The string operator is the concatenation symbol designated by double exclamation marks (!!). Concatenation is the joining of one string to another.

As an example,

```
C: X$ = "DOG"
C: Y$ = "CAT"
C: Z$ = X$!!" & "!!Y$
```

results in Z$ = "DOG & CAT"

5.9.12 Edit Options

The Compute instruction can also be used to perform editing on string variables.

The edit options are as follows.

Symbol	Meaning	Example
U	Translate entire string to upper-case.	C: Z$ = "Ducklings" then C: /Z$ U results in Z$ = DUCKLINGS
C	Capitalize the first character in string.	C: Z$ = "north" then C: /Z$ C results in Z$ = North
xy	Replace all x characters with y characters (x and y can be any character.	C: Z$ = "killed" then C: /Z$ /ld results in Z$ = kidded.
x	Remove all occurrences of x (x can be any character).	C: Z$ = "2,500" then C: /Z$ /, results in Z$ = 2500

5.9.13 Functions

Arithmetic and string functions can be included in an expression. The available functions are ABS, ATN, COS, EXP, FIX, INT, LOG, LN, RND, SGN, SIN, SQR, ASC, CHR, FLO, INS, LEN, and STR.

See section 8 for a description of these functions.

5.9.14 Operator Precedence

When the expression in a Compute instruction contains more than one operator, a set of rules determines the order in which the expression is evaluated. The rules are as follows.

1. Sub-expressions in the innermost sets of parentheses are evaluated first. These values then become operands for the expression containing those sets of parentheses.

2. When the order of evaluation is not determined by parentheses, the operations are performed according to the precedence shown in the following table.

Operator	Order
tilde	Performed first
**	Performed next
* /	Performed next
+ - !!	Performed next
= < > <> <= >=	Performed next
! &	Performed last

3. Operations of equal precedence are performed from left to right. Exponentiation is the exception to this. It is performed from right to left. For example, in the expression 8**5**2 the value 5 is first raised to the power of 2, and then 8 is raised to the 25th power.

5.9.15 Examples

Here are some examples of how expressions are evaluated.

C: X = 12/3+1	results in X = 5.
C: X = 12/(3+1)	results in X = 3.
C: X = 12-(3>1)	results in X = 11; The value of the relational expression (3>1) is 1 (the true condition).
C: X = (12/(4-(3>1)))**2	results in X = 16. The innermost set of parentheses (3>1) is evaluated first. Its value is 1 (the true condition). The next outer set is evaluated next (4-(3>1)). Its value is 4 minus 1, or 3. The outermost set of parentheses is evaluated next (12/(4-(3>1))). Its value is 12 divided by 3, or 4. Finally, this intermediate result is raised to the 2nd power (4 ** 2), or 16.

5.10 Dimension—D:

Format

D: numeric-array-name(integer[,integer])[; . . .]
D: string-variable(integer)[; . . .]

The Dimension instruction reserves space for numeric-arrays and for string-variables. More than one variable can be defined by an individual Dimension instruction by separating each variable entry with a semicolon (;).

A numeric array or a string variable must have space reserved by a Dimension instruction before it is used elsewhere in a program. It is normally good programming practice to put all Dimension instructions at or very near the beginning of the program.

A numeric array can have one or two dimensions. The values for the dimensions of a numeric array can range from 0 through 400 (maximum). A string variable can have one dimension. The values for the dimension of a string variable can range from 0 through 999 (maximum).

TI PILOT allows storage of up to 400 numeric values and 2000 single character values. However, this space is shared by the PILOT interpreter for scratch-pad use of temporary variables. The Workspace instruction returns the available memory space. (See section 5.17--Workspace.)

Though a string value can be as long as 999 characters, the maximum number of characters for a string variable that can be stored on a disk record is 127. Also, the screen displays only 80 characters of a string value. If a string value exceeds 80 characters, the 80th character displayed is replaced by the last character in the string. Therefore, it is best to limit the length of a string variable to a maximum of 80 characters.

Examples

```
D: Z5(10,7)      (defines a numeric array called Z5 with 10 rows and 7
                  columns.)

D: Z$(18)        defines a string variable with a maximum length of 18
                  characters.
```

5.11 Execute Indirect--XI:

Format

```
XI: string-variable
```

The Execute Indirect instruction causes the performance of an instruction which has been constructed in a string-variable. The string variable must contain characters which compose a legitimate TI PILOT instruction. The instruction in the string variable cannot have a label preceding it.

Example

Suppose that a program has ten kinds of exercises and each exercise group begins with the label EXERx, where x is a number from 1 through 10. The student could be allowed to choose an exercise group as follows.

```
D: X$(2);Z$(9)
C: X = 0
T: CHOOSE AN EXERCISE GROUP BY ENTERING A NUMBER FROM
 : 1 THROUGH 10.
A: #X
TE(X  < 1 ! X > 10): ENTER A NUMBER FROM 1 THROUGH 10.
JEC: @A
C: X$ = STR(X)
C: Z$ = "J: EXER"!!X$
XI: Z$
```

The first two instructions reserve space for the string variables X$ and Z$ and the third instruction defines the numeric variable X. The fourth and fifth instructions (a Type instruction and a Type continuation) prompt the student.

The sixth instruction accepts the student's response and the number entered by the student is assigned to the numeric variable X. If the response does not include a number (the E conditioner is set) or if the number entered is not in the required range, the seventh instruction displays a message to remind the student to enter a number from 1 through 10.

If the student response is not acceptable, the eighth instruction causes a jump back to the Accept instruction. If the student response is acceptable, the ninth instruction converts the number entered into a string value called X$, and the tenth instruction concatenates this string value with the characters J: EXER to form an instruction in the string variable Z$. The eleventh instruction then executes the instruction which has been composed in Z$.

5.12 File Output--FO:

Format:

FO: record-number,string-variable

The File Output instruction lets you write a record to a file on a diskette. You can use this instruction to save data such as student replies, test scores, performance statistics, or information relevant to a program on a diskette.

Before a file can be opened, space for the string variable must be reserved with a Dimension instruction, and the variable must be defined with a Compute instruction. The Dimension instruction specifies the number of characters in the string variable (up to 15 characters are permissible for a filename). The Compute instruction assigns the filename to the string-variable.

The first FO: instruction in a program opens the file. The file must be opened before any other File instructions are done. Otherwise the computer will not know what file to use. The record number used in this FO: instruction sets the maximum number of records for the file. This number can range up to 695 if all of the space on the diskette is unused. See the Filer manual for more information on disk structures.

The example below illustrates the instructions necessary to open a file.

```
D: G$(15)                 Reserves space for the string variable used to
                          contain the filename.
C: G$ = "MYFILENAME"      Assigns the filename to the string variable G$.
FO: 35,G$                 Opens file called MYFILENAME with 35 records
                          available.
```

Note that the filename must adhere to the UCSD p-System syntax. The above example assigns the full file name of 4:MYFILENAME (4 is the unit number of disk drive 1). If a prefix volume had been designated by using the Filer (for example, "PILOT:"), then the file would be under the filename PILOT:MYFILENAME. Refer to the Filer manual for more information.

In subsequent FO: instructions, the record number is an integer constant or a numeric variable which specifies the individual record that is written (695 possible records on one diskette). The record number ranges from zero to the size of the file. A record is 128 bytes long (the 128th byte is always a carriage-return character). The data in the string-variable are written to the record on diskette.

The data are padded on the right with spaces to a length of 127, if necessary. Also, if the data exceed the maximum length of the string-variable, the data are truncated (deleted) on the right.

The following is an example of the use of an FO: instruction after a file has been opened by an earlier FO: instruction.

```
    .
Z$ = "THE AVERAGE SCORE IS 93.6"
    .
    .
FO: 8,Z$
```

In the above example the FO: instruction writes the characters in the string variable Z$ to record number 8 of a file on diskette. The data are padded on the right with enough spaces to make the record length equal to 127.

5.13 File Input--FI:

Format

FI: record-number,string-variable

The File Input instruction lets you read a record from a file on a diskette. You can use this instruction to read data such as student replies, test scores, performance statistics, or information relevant to a program which was previously saved on a diskette.

In order for an FI: instruction to be performed, the file must first be opened by the first FO: instruction in the program. Otherwise the computer will not know what file to read from. (See section 5.12--File Output).

The record-number is an integer constant or a numeric variable which specifies the individual record which is read. The record number ranges from zero to the size of the file. A record is 128 bytes long. The data in the record are read from the diskette and placed in the string variable.

The following example illustrates the FI: instruction.

```
D: Z$(256)
     .
     .
     .
FI: 67,Z$
```

The Dimension statement sets the maximum size of the string variable Z$ to 256 characters, and then the FI: instruction reads record number 67 from a diskette file into Z$. However, only 127 characters are read. This is because 127 is the length of a record, and the FI: and FO: instructions will not cross record boundaries.

5.14 Graphics--G:

Format

```
G: command-list
```

The Graphics instruction causes pictorial information to be displayed. The Graphics instruction lets you take advantage of the advanced graphics capability of the TI Home Computer.

The command-list contains one or more of the following commands. Each command is separated by a semicolon (;) and the individual parameters of a command (when specified) are separated by a comma.

T — Erases screen and sets screen to text mode. In text mode, the display is 40 columns by 24 lines and you cannot use sprites.

P — Erases screen and sets screen to pattern mode. In pattern mode, the display is 32 columns by 24 lines and you can use sprites.

Fn — Sets foreground color to n where n is a number from 0 through 15. (See Appendix D for a list of the colors and their numbers.)

Bn — Sets background color to n where n is a number from 0 through 15. (See Appendix D for a list of the colors and their numbers.)

Cn,f,b — Sets character number n (where n is a number from 0 through 255), foreground color to f (where f is a number from 0 through 15), and background color to b (where b is a number from 0 through 15. See Appendix C for character-set groupings.

X0 — Sets sprite magnification to single. This means that each character in a sprite occupies just one character position on the screen.

X1 — Sets sprite magnification to double. This means that each character in a sprite takes up four character positions on the screen. Each dot position in the character expands to occupy four dot positions on the screen. The expansion is from a single magnification and is down and to the right.

X2 — Sets sprite size to single. This means that each sprite is defined by one character.

X3 — Sets sprite size to double. This means that each sprite is defined by four characters that include the character specified. The first character is the one specified when the sprite was created if its number is evenly divisible by four or the next smallest number that is evenly divisible by 4. That character is the upper-left quarter of the sprite. The next quarter is the lower-left quarter of the sprite. The next character is the upper-right quarter of the sprite. The final character is the lower-right quarter of the sprite. The character specified when the sprite was created is one of the four that makes up the sprite.

Mx,y — Moves cursor to column x (where x is from 0 through 31 in pattern mode and 0 through 39 in text mode) and row y (where y is from 0 through 23).

Wp,x-position,y-position,r

The W command horizontally displays the pattern p (where p is from 0 through 255 and defined by the loaded pattern file) starting at position x,y (where x-position is 0 through 31 in pattern mode and 0 through 39 in text mode, and y-position is 0 through 23) for r repetitions. When the maximum x-position is exceeded and there are still repetitions remaining, the pattern is displayed on the next row down (y-position + 1) at x-position 0.

Vp,x-position,y-position,r

The V command vertically displays the pattern p (same as above) starting at postion x,y (same as above) for r repetitions. When the maximum y-position is exceeded and there are still repetitions remaining, the pattern is dziplayed on column to the right (x-position + 1) at y-position 0.

Sn,p,c,x-position,y-position,x-velocity,y-velocity

The S command creates sprites. Sprites are graphics characters which have a color and a location anywhere on the screen. They can be set in motion in any direction at a variety of speeds, and continue their motion until it is changed by the program or the program stops. They move more smoothly than the usual character which jumps from one screen position to another.

The command starts sprite n (where n is from 0 through 31) with pattern p (where p is from 0 through 255), color c (where c is from 0 through 15), x-position (where x-position is from 0 through 255) and y-position (where y-position is from 0 through 255), and with x-velocity (where x-velocity is from -128 through 127) and y-velocity (where y-velocity is from -128 through 127).

n is a numeric value from 1 through 28. If the value is that of a sprite already defined, the old sprite is deleted and replaced by the new sprite. If the old sprite has a row- or column-velocity, and no new one is specified, the new sprite retains the old velocities.

Sprites pass over fixed characters on the screen. When two or more sprites are coincident, the sprite with the lowest sprite number covers the other sprites. While five or more sprites are on the same screen row, the one(s) with the highest sprite number(s) disappear.

p may be any integer from 10 through 255 and designates the character pattern. The character pattern can be defined by the PATTERN subprogram. The sprite is defined as the character given, and in the case of double-sized sprites, the next three characters.

c may be any numeric value from 0 through 15. It determines the foreground color of the sprite. The background color of a sprite is always 1, transparent.

x-position and y-position set the position of the sprite on the screen. They are numbered consecutively starting with 0 in the upper-left-hand corner of the screen. x-position can be from 0 through 255, and y-position can be from 0 through 191. Actually, y-position can be as large as 255, but the positions from 192 through 255 are off the bottom of the screen. (Each character is generated by an 8 X 8 dot matrix, making 255 positions possible for either x- or y-positions.) The position of the sprite is the upper-left-hand corner of the character(s) which defines it.

x-velocity and y-velocity may optionally be specified when the sprite is created. If both x-velocity and y-velocity are zero, the sprite is stationary. A positive y-velocity moves the sprite down and a negative y-velocity moves it up. A positive x-velocity moves the sprite to the right and a negative value moves it to the left. If both x-velocity and y-velocity are non-zero, the sprite moves at an angle in a direction determined by the actual values.

x-velocity and y-velocity can be from -128 through 127. A value close to zero is very slow. A value far from zero is very fast. When a sprite comes to the edge of the screen, it disappears and reappears in the corresponding position on the other side of the screen.

Hn Halts sprite n. Sprite disappears from screen.

D Restores default character set.

5.15 Sound—S:

Format

S: v,command-list

The Sound instruction lets you send sound to the student.

The instruction starts the processing of a sound list for the voice v (where v is from 1 through 4). Voices 1, 2, and 3 are for tones and notes. Voice 4 is for periodic noise and white noise.

There is an important difference between tones and notes. A tone sounds for the entire duration specified; a note sounds for 7/8 of the duration specified and is silent for the remaining 1/8 of the duration. Thus, a series of tones sounds connected or legato, and a series of notes sounds separated or staccato.

Each command in a command-list (except for the last one) is followed by a semicolon. The semicolon is optional after the last command in a command-list. The individual parameters of a command (when specified) are separated by a comma. The command-list consists of one or more of the following commands.

C	Clears the sound list for voice v.
H	Halts sound v.
Vn	Adds a volume command to the sound list, where n is a number from 0 through 15 (0 is silence and 15 is the loudest volume). Uses 2 bytes.
Tp,d	Adds a tone command to the sound list, with pitch of p (110...16383) and duration of d (1...16). Uses 3 bytes.
Sn	Sets tempo (speed) of voice v to n (1...32767). The tempo n is the duration of one beat in milliseconds. Uses 1 byte.
Np,d	Adds a note command to the sound list, where pitch is p (110...16383), and duration is d (1...16). Uses 3 bytes.
Wt,d	Adds a white noise to the sound list (only with voice 4) of type t (0...3) and duration, d (1...16). Uses 3 bytes.
Yt,d	Adds a periodic noise to the sound list (only with voice 4) of type t (0...3) and duration, d (1...16). Uses 3 bytes.
P	Ends sound list for voice v; plays the sound list. Uses 1 byte.

It is necessary to clear the sound list (using the C command) before playing the next sound list.

Sound lists built by using the above commands require memory space, the amount of which varies according to the number of and types of commands used. Up to 400 bytes of memory can be used for each voice. Note that the amount of memory (measured in bytes) required by each command is given at the end of the description.

An extremely long sound list could exceed the 400 byte limit. To determine if a given sound list does exceed this limit, add the total number of bytes used by all of the commands in the sound list.

Also note that if a program contains a very long sound list followed by another sound list, a disk operation to access commands in the subsequent sound list could cut short or otherwise interfere with the playing of the first sound list. If this occurs, use a counting loop between sound lists to delay further PILOT interpretations so the first sound list can complete playing. Below is a sample counting loop.

```
C: X=0
*LOOP
C: X=X+1
J (X<15): LOOP
```

It is necessary to clear the sound list for a voice (using the C command) before making and playing another sound list.

5.16 Speech--V:

Format

V: string-variable

The Speech instruction allows you to send speech to the student. NOTE: The Solid State Speech™ Synthesizer is required for this instruction to function.

The computer speaks the string of words contained in the string-variable following the V: instruction. Space for the string-variable must be reserved (by a D: instruction) and the variable must be defined (by a C: instruction) for the V: instruction to work. The example below illustrates this.

```
D: A$(25)
C: A$="HELLO THERE"
V: A$                          (Computer speaks the words contained in the
                               string variable A$.)
```

5.17 Workspace--W:

The Workspace instruction returns the current locations of the numeric buffer pointer and the character buffer pointer. This instruction also returns the number of locations available for additional string and numeric values. The screen displays:

```
CURRENT WORKSPACE STATUS
The character buffer pointer
is at--###.
There are ### locations available
for string variables and scratch pad use.
The numeric buffer pointer
is at--###.
There are ### locations available
for numeric values and scratch pad use.
```

The number of available locations given is the current amount of unused memory left in the buffers. This available memory can be used for programs and by the system (scratch pad area).

SECTION 6: STUDENT COMMANDS

There are two commands which can be entered by the student while a PILOT program is running. Either one of the commands may be entered by the student in conjunction with an Accept instruction. Each of the individual commands may be enabled or disabled by an option code used with the PR: instruction (see section 5.1 for a description of the Problem instruction). If a PR: instruction has enabled a command, the requested action is performed. If the command is disabled, a command is treated just as any other response.

6.1 GOTO Command

Format

    GOTO destination

The GOTO command lets a student initiate a jump from the current location to another place in the program. You would normally tell the student during the course of the program of this capability and describe to the student the possible destinations. You used the GOTO command with the DEMO program included on the diskette.

The destination may be any of the choices available with the Jump instruction. Normally, the destination is a label. Occasionally, a @P destination may be useful.

You may find the GOTO command useful also when you are testing your program. You can jump from one section of your program to another to exit a loop or to bypass a section of the program which either has a bug in it or is already known to be correct.

As an example, in response to an Accept instruction, a student may enter

    GOTO PROBS

6.2 Escape Command

Format

    @ text

The Escape command lets a student call a special routine which has been built into the program.

When the Escape command is enabled by a PR: instruction, TI PILOT examines the first character in each student response, looking for a "@" character. If one is found, TI PILOT automatically performs the instruction, "U: SYSX". There must be a subroutine named SYSX in the program to receive control. The subroutine can scan the text entered by the student and perform the desired function.

The return from the SYSX subroutine must be via an E: instruction. The return is back to the instruction following the Accept instruction unless a label or other destination is included in the End instruction (see section 5.8 for a description of the E: instruction).

You can use the Escape command to build custom execution time commands. For example, a student (or author) could be allowed to build a TI PILOT instruction as follows.

```
    D: R$(40);Z$(50)          (It is good programming practice to reserve space
                               for all string variables at the beginning of your
                               program.)
    .
    .
    .
*SYSX
    C: R$ = %B
    C: Z$  @  U
    XI: Z$
    E:
```

If, in response to an Accept instruction, the student should enter @T: #N, this routine would display the value of the numeric variable N and then return control to the instruction following the Accept instruction.

SECTION 7: FUNCTIONS

7.1 Numeric Functions

Here is a list of each of the TI PILOT numeric functions, the definition of the function, and an example of the function.

Function	Definition	Example
ABS(N)	Absolute value of N	ABS(-8.7) = 8.7 ABS(6) = 6
ATN(N)	Arctangent of N (in radians)	ATN(0) = 0
COS(N)	Cosine of N (in radians)	COS(3.14) = -1
EXP(N)	e ** N	EXP(1) = 2.71828... EXP(3.8) = 44.701
FIX(N)	Truncate N	FIX(10.8) = 10 FIX(-10.8) = -10
INT(N)	Integer of N	INT(10.8) = 10 INT(-10.8) = -11
LOG(N)	Base 10 logarithm of N	LOG(100) = 2 LOG(220) = 5.3936...
LN(N)	Base e logarithm of N	LN(10) = 2.3026... LN(7) = 1.9459
RND(N)	Random Number If N < 1, RND(N) = a number from 0 to <1 (a fractional number).	RND(0.8) = 0 to .99...
	If N > or = 1, RND(N) = an integer from 0 to N-1.	RND(6) = 0,1,2,3,4, or 5
SGN(N)	Sign of N: -1 for negative N, 0 for N = 0, 1 for positive N	SGN(-8.6) = -1 SGN(4-4) = 0 SGN(56E-7) = 1
SIN(N)	Sine of N (in radians)	SIN(3.14/2) = 1
SQR(N)	Square root of N	SQR(25) = 5

7.2 String Functions

Here is a list of each of the TI PILOT string functions, the definition of the function, and an example of the function.

Function	Definition	Example
ASC(R$)	The ASCII code value of the first character in R$ (See the ASCII character code in Appendix B)	C: S$ = "HORSES" C: X = ASC(S$) causes X = 72
CHR(N)	The character whose ASCII character code is N	C: S$ = CHR(10) causes a line feed character to be assigned to S$.
FLO(R$)	The numeric value of the first number in R$. The numeric value is in floating point format.	C: S$ = "IT IS 8 BELOW" C: N = FLO(S$) causes N = 8
INS(N,Q$,R$)	A number which is the starting position of the string Q$ within the string R$. The search is started in the Nth position of string R$. If the characters in Q$ are not found within R$, N is set to zero. Q$ and R$ can not total a length greater than 255.	C: S$ = "KITTENS" C: Y$ = "TEN" C: N = INS(2,Y$,S$) causes N = 4
LEN(R$)	A number which is the current length of R$.	C: S$ = "KITTENS" C: N = LEN(S$) causes N = 7

STR(N)	A conversion of N to a string. The resulting characters in the string will express a decimal format if the number is in the limits of a decimal format; otherwise, the string will express a floating point format.	C: X = 86.5 C: S$ = STR(X) results in S$ being assigned the characters 8, 6, ., and 5

SECTION 8: PROGRAMMING RECOMMENDATIONS

This manual introduces you to the TI PILOT language and helps you learn the language so that you can write your own programs. In developing programs using TI PILOT, there are techniques that you can use which will improve the effectiveness and efficiency of those programs. This section contains some recommendations that you may find useful.

8.1 Planning

You will save time by carefully planning your program before you begin writing it. Define the purpose of the program and determine the sequence of topics to be presented.

Look for opportunities to improve the effectiveness of your program by using the graphics and sound features of the TI Home Computer; for example, use graphs and animation where they are suitable. You will find it helpful to design the screen displays before beginning to write the program. Use quadriruled paper to draw the graphics display and to plan the text displays that will appear on the screen.

8.2 Program Structures

As you begin composing your program, use a top-down design process; that is, first define the program functions in general and then define each function in more detail. Divide the functions into separate program modules, and as you write the program, start with the most general program modules and then develop the more detailed program modules.

Strive for a simple program structure; that is, use the most straightforward sequence of instructions you can.

As an example, the following program segment displays one of eight directions corresponding to a compass heading. A heading of 337.5 degrees to 0 degrees and 0 degrees to 22.5 degrees produces a direction of North; a heading of 22.5 degrees to 67.5 degrees produces a direction of Northeast; a heading of 67.5 degrees to 112.5 degrees produces a direction of East; and so forth.

```
R:
R: H IS A NUMERIC VARIABLE CONTAINING A HEADING
R:
     T(((H >= 337.5) & (H < 360)) ! ((H >= 0) & (H < 22.5))): NORTH
     T((H >= 202.5) & (H < 247.5)): SOUTHWEST
     T((H >= 67.5) & (H < 112.5)): EAST
     T(H >= 360): HEADING IS TOO LARGE
     T((H >= 292.5) & (H < 337.5)): NORTHWEST
     T((H >= 22.5) & (H < 67.5)): NORTHEAST
     T((H >= 157.5) & (H < 202.5)): SOUTH
     T((H >= 247.5) & (H < 292.5)): WEST
     T(H < 0): HEADING IS TOO SMALL
     T((H >= 112.5) & (H < 157.5)): SOUTHWEST
```

Although the program will work, the following program segment is more straightforward.

```
    T(H < 0): HEADING IS TOO SMALL
    T(((H >= 337.5) & (H < 360))!((H >= 0) & (H < 22.5))): NORTH
    T((H >= 22.5) & (H < 67.5)): NORTHEAST
    T((H >= 67.5) & (H < 112.5)): EAST
    T((H >= 112.5) & (H < 157.5)): SOUTHEAST
    T((H >= 157.5) & (H < 202.5)): SOUTH
    T((H >= 202.5) & (H < 247.5)): SOUTHWEST
    T((H >= 247.5) & (H < 292.5)): WEST
    T((H >= 292.5) & (H < 337.5)): NORTHWEST
    T( H >= 360): HEADING IS TOO LARGE
```

Within a program, group all of the variable definition statements at the beginning of the program. Put all of the Compute statements together that define numeric variables and put all of the Dimension statements together defining numeric arrays and string variables. Use Remark statements to describe these variables as they are defined at the beginning of the program.

For example, the following program segment defines the program variables.

```
R:
R: THE FOLLOWING NUMERIC VARIABLES ARE USED:
R:
R:       N - NUMBER OF QUESTIONS
R:       C - NUMBER OF CORRECT ANSWERS
R:       P - PERCENTAGE OF CORRECT ANSWERS
R:
    C: N = 0
    C: C = 0
    C: P = 0
R:
R: THE FOLLOWING STRING VARIABLES ARE USED:
R:
R:      A$ = STUDENT'S NAME
R:
    D: A$(30)
R:
```

8.3 Subroutines

Very often you will find it useful to create a subroutine to perform a specific program function.

When you create a subroutine, you are less likely to introduce "bugs" in your program by constructing the subroutine with only one entry point and one exit point; that is, the subroutine should be constructed to receive control from a calling program at only one point (one specific label) and when the subroutine returns control to a calling program, it should do so from only one point.

Here is an example of a program segment that includes two calling programs and a subroutine that has more than one entry point and more than one exit point.

```
          .
          .
          .
R: THIS IS A CALLING PROGRAM.

     U:SUBA                                   (Call the subroutine at
                                               one entry point)
          .
          .
          .
R: THIS IS ANOTHER CALLING PROGRAM.

     U: SUBB                                  (Call the subroutine at a
                                               second entry point)
          .
          .
          .
```

```
R: THIS IS THE SUBROUTINE.

*SUBA                                          (one entry point)

     C: Z = X * Y
     J: CHECK

*SUBB                    (a second entry point)

     C: Z = X / Y

*CHECK

     J(Z < 0): NEG
     E:                                        (one exit point)

*NEG

     C: Z = ABS(Z)
     E:   (a second exit point)
          .
          .
          .
```

The program can be reconstructed so that it is less likely to cause problems by creating two subroutines where each subroutine has only one entry point and one exit point.

```
          .
          .
          .

R: THIS IS A CALLING PROGRAM

     U: SUBA                                (calls the first subroutine)
          .
          .
          .
     U: SUBB                                (calls the second subroutine)
          .
          .
          .

R: THIS SUBROUTINE MULTIPLIES X TIMES Y.

*SUBA                                       (Subroutine entry point)

     C: Z = X * Y
     J( Z >= 0): SUBAE
     C: Z = ABS(Z)

*SUBAE

     E:                                     (Subroutine exit point)
          .
          .
          .

R: THIS SUBROUTINE DIVIDES X BY Y.

*SUBB                                       (Subroutine entry point)

     C: Z = X / Y
     J( Z <= 0): SUBBE
     C: Z = ABS(Z)

*SUBBE                                      (Subroutine exit point)

     E:
          .
          .
```

By dividing different functions within your program into separate subroutines, you can create a "library" of program modules that you can move easily from one program to another.

8.4 Documentation

Often, when a program is first written (and sometimes even after it has been used for a while), you will discover a mistake which needs to be corrected, or you will want to make some changes. Discovering mistakes and changing a program are easier if the program is well documented. The following are some suggestions regarding documentation.

Make use of Remark statements in the program to describe the variables used and to describe the processes in the program. It is useful to provide a narrative describing the overall program structure at the beginning of the program.

Use spaces within the statements (where they are allowed) to improve the readibility of the program. For example,

```
C: X = (N/12) ** 3
```

is easier to read than

```
C:X=(N/12)**3
```

It will also improve the readability of your program to use a separate line for labels and to offset operation codes from the left-hand margin. For example,

```
*LABEL
     C: X = (N/12) ** 3
```

is easier to read than

```
*LABEL  C: X = (N/12) ** 3
```

Note that though the use of Remark statements and extra spaces is a good programming practice, they are nonessential to the computer's performance of a program. Remarks and extra spaces do occupy memory and, if overused, may cause the program to run somewhat slower.

Use labels that reflect the purpose of the program segment associated with the label. For example, instead of using a label like X1 for a program segment that averages numbers, use a label like AVERAG.

As you begin to create several programs, you may want to develop a documentation file for the programs. Use a file folder for each program and include in the folder a print-out of the statements in the program, a program narrative, a history of any revisions to the program, and a description of the results of running the program.

8.5 Increasing Program Effectiveness

There are several techniques you can use to improve the effectiveness of the programs you write. A few techniques are described below.

8.5.1 Use Personal Names

You can use a string variable with the Accept instruction and the Type instruction to personalize a program for a student. For example, you can use the Accept instruction to learn the student's name.

```
T: WHAT IS YOUR NAME?
A: N$
```

You can then use the student's name whenever it is appropriate. For example,

```
T: OK, $N$ , TODAY WE WILL STUDY ADVERBS.
```

8.5.2 Use Variety

You can use the Random (RND) function to add variety to a program. For example, instead of giving the same response each time to a correct answer, give different responses. The following program segment shows you one way to do this.

```
T: WHEN WAS THE DECLARATION OF INDEPENDENCE SIGNED?
A:
M: 1776
TN: NO, TRY AGAIN
JN: @A
U: CORANS
                    .
                    .
                    .
*CORANS
     C: X = RND(3) + 1
     J( X = 3): THREE
     J( X = 2): TWO
R:
```

```
R: THE NEXT TWO INSTRUCTIONS ARE PERFORMED IF X = 1
R:
     T: GOOD.
     J: RETURN
R:
R: THE NEXT TWO INSTRUCTIONS ARE PERFORMED IF X = 2
R:
*TWO
     T: THAT'S CORRECT.
     J: RETURN
R:
R: THE NEXT INSTRUCTION IS PERFORMED IF X = 3
R:
*THREE
     T: YOU GOT IT RIGHT!
*RETURN
     E:
          .
          .
          .
```

In this example, the first instruction poses the question. The second instruction accepts the student's response. The third instruction matches the response with the correct answer. If the response is not the correct answer, the fourth and fifth instructions display the message NO, TRY AGAIN and awaits another response.

If the response is correct, the sixth instruction calls the subroutine CORANS. The subroutine assigns a random integer from 1 through 3 to the numeric variable X. The RND function in the first instruction of the subroutine produces a random number of 0, 1, or 2, and 1 is added to the random number to make X equal to 1, 2, or 3.

8.5.3 Use the XI: Instruction to Experiment with Other Instructions

As you develop a program, you may find it helpful to use the Execute Indirect (XI:) instruction to experiment with other instructions. For example, the following is a short program which will let you experiment with the Graphics instruction.

```
PR:
R:
R:   THIS PROGRAM ACCEPTS CHARACTERS ENTERED FROM THE KEYBOARD
R:   AND USES THESE CHARACTERS AS A COMMAND-LIST FOR A
R:   GRAPHICS INSTRUCTION.  THE GRAPHICS INSTRUCTION WITH
R:   THE COMMAND-LIST WHICH WAS ENTERED IS PERFORMED BY USING
R:   AN XI: INSTRUCTION.
R:
R:   THE PROGRAM PERFORMS EACH G: COMMAND AND THEN LOOPS BACK
R:   TO ACCEPT ANOTHER INPUT FOR ANOTHER G: INSTRUCTION.  THE
R:   PROGRAM REMAINS IN THIS LOOP UNTIL THE WORD QUIT IS
R:   ENTERED.  AT THAT TIME, THE PROGRAM ENDS AND RETURNS TO
R:   THE P-SYSTEM.
R:
R:   THE FOLLOWING STRING VARIABLES ARE USED:
R:
R:       G$--USED TO HOLD THE INSTRUCTION PERFORMED BY
R:           THE XI: INSTRUCTION.
R:       A$--USED TO HOLD THE COMMAND-LIST WHICH IS
R:            ENTERED FROM THE KEYBOARD.
R:
     D: G$(80)
     D: A$(78)
*ACCEPT
     A: $A$
     M: QUIT
     JY: END
     C: G$ = "G:"!!A$
     XI: G$
     J: ACCEPT
*END
     E:
```

1. Be sure that the diskette you are using is the correct one. Use the L(ist-directory) command in the Filer to check for the correct diskette or program.

2. Ensure that your Memory Expansion unit, P-Code peripheral, and Disk System are properly connected and turned on. Be certain that you have turned on all peripheral units and have inserted the appropriate diskette before you turn on the computer.

3. If your program does not appear to be working correctly, end the session and remove the diskette from the disk drive. Insert the diskette again, and follow the "Getting Started" instructions carefully. If the program still does not appear to be working properly, remove the diskette from the disk drive, turn the computer and all peripherals off, wait 10 seconds, and turn them on again in the order described above. Then load the program again.

4. If you are having difficulty in operating your computer or are receiving error messages, refer to the "Maintenance and Service Information" and "Error Messages" appendices in the User's Reference Guide or P-Code Card manual for additional help.

5. If you continue to have difficulty with your computer or with TI PILOT, please contact the dealer from whom you purchased the unit or program for service directions.

APPENDIX A. LANGUAGE SUMMARY

The following is a summary of the TI PILOT language syntax, including a list of the operation codes in alphabetical order, a list of the modifiers and conditioners, a list of the Compute instruction operators, a concise description of the command-lists used with Graphics and Sound instructions, and a summary of other features of the language.

SYNTAX:

*label op-code modifier(s) conditioner(s) (relational-expression): text-field

LABELS

Must begin with an *
One through six alphanumeric characters
First character must be alphabetic
May be on a separate line preceding the op-code

OPERATION CODES (op-codes)

R: Remark

T: Type

A: Accept

U: Use

E: End

C: Compute

D: Dimension

XI: Execute Indirect

FO: File Output

FI: File Input

G: Graphics

S: Sound

V: Speech

J: Jump

    Targets for a jump instruction:
    <label>   (the instruction at <label>)
    @A   (the last Accept instruction)
    @M   (the next Match instruction)
    @P   (the next Problem instruction)

M: Match
    Special matching features:

    &   match one word and another
    !   match one word or another
    %   match for a space or the start or end of an answer
    *   match any single character

PR: Problem
    Option list:

    E   escape allowed
    G   GOTO allowed
    L   convert input to lower-case in Answer Buffer
    S   remove spaces from student's input in Answer Buffer
    U   convert input to upper-case in Answer Buffer
    W   clear labels
    B   clear variables

W: Workspace Status

MODIFIERS

H	suppress line feed with Type (TH:)
J	automatic jump with Match (MJ:)
S	spelling correction with Match (MS:)
X	suppress input text editing with Accept (AX:)

CONDITIONERS

C	perform if last relational expression was true
E	perform if Error condition is set
N	perform if No match
Y	perform if Yes match
n	perform if %A equals n

RELATIONAL-EXPRESSION

Enclosed in parentheses
Instruction performed only if expression is true.
If expression is true, C conditioner is set.

TEXT-FIELD

Used by individual instructions for constants and variables.

STUDENT COMMANDS

GOTO label	Jump to instruction at label.
@ text	Escape to SYSX subroutine. Text, if any, is in %3.

COMPUTATIONAL FEATURES

The following operators and options can be used with the Compute instruction.

ARITHMETIC OPERATORS

**	exponentiation
*	multiplication
/	division
+	positive value or addition
-	negative value or subtraction

RELATIONAL OPERATORS

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

LOGICAL OPERATORS

	not
&	and
!	or

STRING OPERATION

!!	concatenation

ORDER OF OPERATIONS

tilde	Performed first
**	Performed next
* /	Performed next
+ - !!	Performed next
= < > <> <= >=	Performed next
! &	Performed last

EDIT OPERATIONS

C	capitalize first letter
U	translate string to upper case
xy	replace x characters with y characters
x	remove all x characters

SYSTEM VARIABLES

%A	Answer Counter
%B	Answer Buffer

SUBSCRIPTING

array-name(c)	Single-dimension array--c is number of columns.
array-name(r,c)	Double-dimension array--r is number of rows and c is number of columns.
string-name(p)	p is position of character in string-name starting from 1.
string-name(p,l)	p is position of character in string-name starting from 1 and l is length of sub-string.

GRAPHICS COMMANDS

Format of Graphics instruction:

G: command-list

Each command in the command-list is separated by a semicolon (;).

T	Erases screen and set to text mode.
P	Erases screen and set to pattern mode.
Fn	Sets foreground color to n (0-15).
Bn	Sets background color to n (0-15).
Cn,f,b	Sets character number n (0-255), foreground color to f (0-15), and background color to b (0-15).
X0	Sets sprite magnification to single.
X1	Sets sprite magnification to double.
X2	Sets sprite size to single.
X3	Sets sprite size to double.
Mx,y	Moves cursor to column position x (0-31 in pattern mode and 0-39 in text mode) and row position y (0-23).
Wp,x,y,r	Displays pattern p horizontally beginning at column position x and row position y for r number of repetitions.

Vp,x,y,r Displays pattern p vertically beginning at column position x and row position y for r number of repetitions.

Sn,p,c,x-position,y-position,x-velocity,y-velocity.

Starts sprite n (0-31) with pattern p (0-255), color c (0-15), x-position (0-255), y-position (0-255), x-velocity (-128 through 127), and y-velocity (-128 through 127).

Hn Halts sprite n (0-31).

D Restores default character set.

SOUND COMMANDS

Format of Sound instruction:

S: v, command-list

v (1-4) is the voice number (1-3 is for tones and notes; 4 is for periodic and white noise).

Each command in the command-list is separated by a semicolon (;).

C Clears the sound list for voice v.

H Halts sound v.

Vn Adds a volume command to the sound list; n (0-15) is the volume. Uses 2 bytes.

Tp,d Adds a tone command to the sound list with pitch p (110-16383) and duration d (1-16). Uses 3 bytes.

Sn Sets tempo (speed) of voice v to n (0-32767). Uses 1 byte.

Np,d Adds a tone command to the sound list with pitch p (110-16383) and duration d (1-16). Uses 3 bytes.

Wt,d Adds a white noise to the sound list of type t (0-3) and duration d (1-16) (only with voice 4). Uses 3 bytes.

Yt,d Adds a periodic noise to the sound list of type t (0-39) and duration d (1-16) (only with voice 4). Uses 3 bytes.

P End of the sound list for voice v; play the sound list. Uses 1 byte.

SPEECH

Format of Speech instruction:

V: string-variable

The default characters on the TI-99/4A Computer are the standard ASCII characters for codes 32 through 127. The following chart lists these characters and their codes.

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
32	(space)	65	A	97	a
33	! (exclamation point)	66	B	98	b
34	" (quote)	67	C	99	c
35	# (number or pound sign)	68	D	100	d
36	$ (dollar)	69	E	101	e
37	% (percent)	70	F	102	f
38	& (ampersand)	71	G	103	g
39	' (apostrophe)	72	H	104	h
40	((open parenthesis)	73	I	105	i
41	) (close parenthesis)	74	J	106	j
42	* (asterisk)	75	K	107	k
43	+ (plus)	76	L	108	l
44	, (comma)	77	M	109	m
45	- (minus)	78	N	110	n
46	. (period)	79	O	111	o
47	/ (slant)	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z
58	: (colon)	91	[(open bracket)	123	(left brace)
59	; (semicolon)	92	(reverse slant)	124	
60	< (less than)	93	](close bracket)	125	(right brace)
61	= (equals)	94	^(caret)	126	(tilde)
62	> (greater than)	95	_(line)	127	DEL(appears on screen as a blank.)
63	? (question mark)	96	(grave)		
64	@ (at sign)				

Two additional characters are predfined on the TI-99/4A Computer. The cursor is assigned to ASCII code 30, and the edge character is assigned to code 31.

APPENDIX C. CHARACTER SETS

These character codes are grouped into 32 sets for use in color graphics programs. NOTE: Changing the foreground and background colors of any one character within a set causes all eight characters within that set to change.

0-7	8-15	16-23	24-31	32-39	40-47	48-55	56-63
64-71	72-79	80-87	88-95	96-103	104-111	112-119	120-127
128-135	136-143	144-151	152-159	160-167	168-175	176-183	184-191
192-199	200-207	208-215	216-223	224-231	232-239	240-247	248-255

APPENDIX D. TI PILOT COLOR CODES

Color	Code #	Color	Code #
Transparent	0	Medium Red	8
Black	1	Light Red	9
Medium Green	2	Dark Yellow	10
Light Green	3	Light Yellow	11
Dark Blue	4	Dark Green	12
Light Blue	5	Magenta	13
Dark Red	6	Gray	14
Cyan	7	White	15

Note: Though these are the same colors used in TI BASIC, the code numbers range from 0 through 15 in TI PILOT. The code numbers in TI BASIC range from 1 through 16.

The following color combinations produce the sharpest, clearest character resolution on the TI-99/4A color screen. The first color listed is the foreground color; the second color listed is the background color. Color codes are included in parentheses.

Black on Medium Green (1, 2)
Black on Light Green (1, 3)
Black on Light Blue (1, 5)
Black on Dark Red (1, 6)
Black on Cyan (1, 7)
Black on Medium Red (1, 8)
Black on Light Red (1, 9)
Black on Dark Yellow (1, 10)
Black on Light Yellow (1, 11)
Black on Dark Green (1, 12)
Black on Magenta (1, 13)
Black on Gray (1, 14)
Black on White (1, 15)
Medium Green on White (2, 15)
Light Green on Black (3, 1)
Light Green on White (3, 15)
Dark Blue on Light Blue (4, 5)
Dark Blue on Gray (4, 14)
Dark Blue on White (4, 15)
Light Blue on Gray (5, 14)
Light Blue on White (5, 15)
Dark Red on Light Yellow (6, 11)
Dark Red on White (6, 15)
Medium Red on Light Red (8, 9)
Medium Red on Light Yellow (8, 11)
Medium Red on White (8, 15)
Light Red on Black (9, 1)
Light Red on Dark Red (9, 6)
Dark Yellow on Black (10, 1)
Light Yellow on Black (11, 1)
Light Yellow on Dark Red (11, 6)
Dark Green on Light Green (12, 3)
Dark Green on Light Yellow (12, 11)
Dark Green on Gray (12, 14)
Dark Green on White (12, 15)
Magenta on Gray (13, 14)
Magenta on White (13, 15)
Gray on Black (14, 1)
Gray on Dark Blue (14, 4)
Gray on Dark Red (14, 6)
Gray on Dark Green (14, 12)
Gray on White (14, 15)
White on Black (15, 1)
White on Medium Green (15, 2)
White on Light Green (15, 3)
White on Dark Blue (15, 4)
White on Light Blue (15, 5)
White on Dark Red (15, 6)
White on Medium Red (15, 8)
White on Light Red (15, 9)
White on Dark Green (15, 12)
White on Magenta (15, 13)
White on Gray (15, 14)

The following table gives frequencies (rounded to integers) of four octaves of the tempered scale (one half-step between notes). While this list does not represent the entire range of tones--or even of musical tones--it can be helpful for musical programming.

Frequency	Note	Frequency	Note
110	A	440	A (above middle C)
117	A♯, B♭	466	A♯, B♭
123	B	494	B
131	C (low C)	523	C (high C)
139	C♯, D♭	554	C♯, D♭
147	D	587	D
156	D♯, E♭	622	D♯, E♭
165	E	659	E
175	F	698	F
185	F♯, G♭	740	F♯, G♭
196	G	784	G
208	G♯, A♭	831	G♯, A♭
220	A (below middle C)	880	A (above high C)
220	A (below middle C)	880	A (above high C)
233	A♯, B♭	932	A♯, B♭
247	B	988	B
262	C(middle C)	1047	C
277	C♯, D♭	1109	C♯, D♭
294	D	1175	D
311	D♯, E♭	1245	D♯, E♭
330	E	1319	E
349	F	1397	F
370	F♯,G♭	1480	F♯,G♭
392	G	1568	G
415	G♯,A♭	1661	G♯,A♭
440	A(above middle C)	1760	A

APPENDIX G. COPYING FILES AND PRINTING WITH THE p-SYSTEM

The following is a decription of how to make a backup copy of p-System disks. Details about the prompts displayed by these programs can be found in the manuals for the Disk Memory System and p-System Editor/Filer/Utilities.

Formatting (Initializing) a New Disk

A new disk to be used for making a copy must be properly formatted for p-System use. This can be done two ways: by using the Disk Manager, which is the easiest way, or by using p-System Filer and Utilities programs.

Using the Disk Manager--Use the Disk Manager to make a backup copy of the disk. The catalog that is displayed shows one file named PASCAL, which uses the entire disk.

Using the p-System method--Format the disk with the DFORMAT program. Load the UTILITY disk in your disk drive and enter

```
UTILITY: DFORMAT
```

in response to the X(ecute command.

Creating a Directory

After the disk is formatted, a directory must be created. This must be done whether you use the Disk Manager or the DFORMAT program to format the disk.

To do this, use the Z(ero command in the FILER program. The disk is now ready for p-System use.

Copying to the New Disk with One Disk Drive (p-System Method)

Use the T(ransfer command in the FILER program. Press I and enter the names of the disk volumes to be used in the transfer process. Type the name of the master disk followed by a colon, a comma, and the name of the copy disk followed by a colon. Note that the copy disk is new disk which was just formatted and Z(eroed. For example:

```
MASTER:,COPY2:
```

Now remove the FILER disk and insert the master disk. Then press <return> in response to the T(ransfer command. The system checks to see if the disk is in the drive, reads some data from it, and then displays the following.

```
How many blocks? 180 (Y/N)
```

This prompt refers to the number of blocks you wish to transfer. Enter Y and the program displays:

```
Put in COPY2:
Type <space> to continue
```

At this point, remove the master disk, insert the copy disk, and press <space>. The data in memory is transferred to the copy disk. Then the program displays:

```
Put in MASTER:
Type <space> to continue
```

Following the prompts on the screen, continue this procedure of alternating the master and copy disks until the all of the data has been transferred.

Copying to the New Disk with Two Disk Drives (p-System Method)

With a two-disk system the procedure is simplified because there is no need to alternate the disks. The T(ransfer command can be answered with

```
#4:,#5:
```

to copy the contents of the disk in unit #4 (Disk Drive 1) onto the one in unit #5 (Disk Drive 2).

Copying Files to a Backup Disk or a Printer

Files can be copied is in much the same way. You only need to add the file name in response to the T(ransfer command.

```
TEST:MYFILE.TEXT,ARCHIVE:TEST1.TEXT
```

Files can also be listed to the printer using the T(ransfer command with the following response.

```
PILOT: DEMO. TEXT,PRINTER:          (or)
PILOT: DEMO. TEXT,REMOUT
```

The PRINTER and REMOUT characteristics must be set to match those of your printer. These characteristics include the baud rate, the number of data bits, and whether your printer is connected to the serial or the parallel output port of the RS232 peripheral.

The p-System can be altered to recognize your printer's parameters by executing the program UTILITY: MODRS232. If, for example, your printer is set to 4800 baud and 8 bits of data and you are connected to serial port #1, then you would execute UTILITY: MODRS232 and press P to change the printer characteristics to:

```
RS232.BA=4800.DA=8.PA=0.EC
```

If you have a TI-99/4 Impact Printer which is connected to port #1 with none of the DIP switches changed, you may use REMOUT: to list files to the printer. If you have changed the baud rate to 4800 and have changed nothing else, you may use the above example by changing the number of data bits to 7 (DA=7).

```
**PR:  UGE
       R: Your programs will be easier to understand
       R: and maintain if the variables are
       R: dimensioned and initialized at the beginning
       R: of the program.

*START D:V1$(80);D9$(1);F9$(1);C$(1);N$(20)

       R: Initialize starting variable values.

 C: D9$=CHR(10)                          : D9$ = move cursor down one line
 C: F9$CHR(137)                          : F9$ = forward space
 : C$=CHR(12)                            : C$ = clear screen
 C: V1$="PLEASE ENTER YOUR FIRST NAME."

       R: Start of program execution for this section.

 T: $C$
 T: WELCOME TO TI PILOT.
 T:
 T: $V1$
 V: V1$                                  :Ask for first name.
 A: $N$
 M: 1!2!3!4!5!6!7!8!9!0 :Did they enter any numbers?

 TY: Are you a computer?  If not, why
   : did you enter numbers?  Try again.

 JY: @A                                  :If they did, then do it again.

 T: $D9$
  : The following "menu" lists the names
  : of program sections within the DEMO
  : program.  To sample one of these
  : sections, type the word GOTO followed by
  : the section name, and then press ENTER.
  : The MENU is displayed again after each
  : section is finished.  You may then
  : sample another section.  When you are
  : ready to leave the DEMO program, type
  : GOTO END and press ENTER.  You are then
  : returned to the p-System promptline.
  :
  : NOTE:
  : To leave the Immediate Mode section
  : (IMMED), you must enter the command
  : GOTO MENU or GOTO END.
  :
  : Do you wish to continue, $N$ ? Y/N
 AS:
 M: Y!y
 JY:MENU
 J: END
```

```
       R:--- M E N U ---

*MENU G: T                                  :set text mode
 PR: UGE
 T: $D9$ $D9$ ************************************
  : *                                      *
  : *             MENU                     *
  : *                                      *
  : *   MATH                SOUND          *
  : *   VERBS               SPEECH         *
  : *   GRAPH               IMMED          *
  : *                                      *
  : *                                      *
  : *            END (to quit)             *
  : *                                      *
  : ************************************
  :
  : $N$ , enter the program section
  : you wish to sample.  Remember, type GOTO
  : followed by the name, and press ENTER.
 A:
 M: GOTO%**
 TN: Remember to type GOTO.
 JN: @A

    R:--- I M M E D I A T E M O D E ---

*IMMED

    R: This program accepts a command from
    R: the screen and immediately executes
    R: it.  This is an excellent way to see
    R: how many of the PILOT commands work
    R: and experiment with them.

*BEGIN
 D: A9$(80);B9$(80)
 C: A9$=CHR(12)                         :clear screen
 T: $A9$ PILOT Immediate mode
 T: Predefined procedures:
  : U: ABC   - dimension a$(256),b$(256),c$(256)
  : <RETURN> - redo last instruction
  :
 C: B9$="T:"

*L1 TH: >
 AX: $A9$
 J(A9$=""): L2
 C: B9$=A9$

*L2 XI: B9$
 J: L1

*ABC D: A$(256)
 D: B$(256)
 D: C$(256)
 E:              : End of subroutine abc.
 J: MENU
```

```
   R:---M A T H D R I L L S ---

*MATH PR: UGE
 T: $C$
  : ***************
  : MATH DRILL #1
  : ***************

 D: T2$(1)T1$(1);T3$(1)
 C: T1$=CHR(12)                    :clear screen
 C: T2$=CHR(08)                    :backspace
 C: T3$=" "                        :space
 D: R1$(7);R2$(32);R3$(32);R4$(5);R5$(32);R6$(35)
 C: R1$="CORRECT"
 C: R2$="NO, THAT IS TOO LARGE.  TRY AGAIN."
 C: R3$="NO, THAT IS TOO SMALL, TRY AGAIN."
 C: R4$="RIGHT!"
 C: R5$="YOU FORGOT TO CARRY.  TRY AGAIN."
 C: R6$="YOU MUST ENTER A NUMBER.  TRY AGAIN."
 C: A=0
 C: N=0
 C: M=0
 C: W=0

   R: Begin the math program.

*GEN J(N-W>4): GEN2
 C: X=RND(9)
 C: Z=RND(7)+2
 C: Y=Z-X
 J(Y<0): GEN
 T: $F9$ #X
 T: +#Y
 T: ----
 TH: $T3$
 A$: $A

   R: Check for erroneous input.

 TE: $R6$
 VE: R6$
 JE: @A

 T:
 T(A=Z): $R1$
 VC: R1$
 C1C: N=N+1
 CC: %A=0
 JC: GEN

 C1: W=W+1
```

```
 T(A-Z): $R2$
 VC: R2$
 JC: @A

 T(A<Z): $R3$
 VC: R3$
 JC: @A

    R: Start Drill #2.

*GEN2
 C: D1=0
 C: D2=0
 C: N=0
 C: W=0

*GEN3 T: $T1$ ********
 T: DRILL #2
 T: ********

 J(N-W> 4): GEN6
 C: X= RND(79)+10
 C: Y= RND(9)
 C: Z=X+Y

*GEN4 T: $T3$ #X
 T: + #Y
 T: ----
 TH: $T3$ $T3$
 AS: #D1

    R: Check for erroneous input.

 TE: $R6$
 VE: R6$
 JE: GEN4

 TH: $T2$ $T2$
 AS: #D2

    R: Check for erroneous input.

 TE: $R6$
 VE: R6$
 JE: GEN4

 T:
 C: A=10*D2+D1
 T(A=Z): $R4$
 VC: R4$
 C1C: N=N+1
 CC: %A=0
 JC: GEN3

 C1: W=W+1
```

```
 T(A+10=Z): $R5$
 JC: GEN4
 T(A>Z): $R2$
 VC: R2$
 JC: GEN4

 T(A<Z): $R3$
 VC: R3$
 JC: GEN4

*GEN6 T:
 T: **** END ****
 J: MENU

    R:--- V E R B S  T E S T ---

*VERBS PR: LGS
 T: $C$
 T:
 T: VERB TEST #1
 T:
 D: L$(15);S$(20);S1$(4);D$(69)
 D: D1$(23);R$(20);F$(20);W$(25)
 D: M1$(7);M2$(7);M3$(8);V$(5);V2$(11)
 C: V$="Right!"
 C: V2$="Not correct."
 C: N=0
 C: S$="I   He   She   You   They"
 C: D$="going to the circus.   coming through the rye."
 C: D$=D$!!"getting saturated.          "
 C: R$="%am%%is%%is%%*re%*re"
 C: F$="%is%%am%%am%x  xx  x"
 C: W$="%*re%%*re%%*re%%*re%am!isam!is"
 C: L$="111111111111111"

*PICK C: X=RND(5)-1
 C: Y=RND(3)-1
 C: I=3*X+Y+1
 J(L$(I,1)="0"): PICK
 C: L$(I,1)="0"
 C: X1=4*X+1
 C: X2=5*X+1
 C: Y1=23*Y+1
 C: S1$=S$(X1,4)
 C: D1$=D$(Y1,23)
 C: M1$= "M: "!!R$(X1,4)
 C: M2$= "M: "!!F$(X1,4)
 C: M3$= "M: "!!W$(X2,5)
```

```
 T:
 T: GIVE THE MISSING VERB.
 T: $S1$         $D1$
 A:
 M: aint!ain*t
 TY: Very funny, now be serious.
 JY: @A
 XI: M1$
 TY: Right !
 VY: V$
 VN: V2$
 CY: N=N+1
 JY(N<6): PICK

 TY:
 TY: **** END ****
 TY:
 JY: FINIS

 XI: M2$
 TY: The subject is singular, but you need
   : the other form of the singular verb.
   : Try Again.
 JY: @A

 XI: M3$
 TY: Think about the number of people that
   : are $D1$ . Try Again.
 JY: @A
 T: The verb must be one of these
  :      are      am      is
  : Try Again.
 J: @A

*FINIS
 J: MENU

    R:--- G R A P H I C S ---

*GRAPH PR: UGE
 D: K$(16)
 C: K$="----------------"

    R: Set up foreground and background colors for the
    R: various character sets.

 G: P;F4;B4;C0;15,1;C32,4,4;
 G: C40,15,1;C48,15,1;C56,15,1;C64,15,1;C72,15,1;C80,15,1;C88,15,1;X2;
 G: C184,14,0;C192,15,0;C200,1,14;C208,5,12;C140,1,12;C148,12,5;
 G: C216,5,12;C224,1,14;C232,1,5;C240,10,0;C160,6,0;C172,6,0;
```

```
   R: Start writing special characters.

G: W192,20,4,2;W195,26,4,1;W197,27,4,1;W198,28,4,1;
G: W192,8,5,1;W195,19,5,1;W193,20,5,2;W197,22,5,1;
G: W198,23,5,1;W189,25,5,1;WW185,26,5,1;W200,27,5,1;
G: W191,28,5,1;W195,7,6,1;W193,8,6,1;W197,9,6,1;
G: W198,10,6,1;W189,18,6,1;W185,19,6,1;W200-20,6,3;
G: W191,23,6,1;W192,24,6,1;W187,25,6,1;W200,26,6,3;
G: W186,29,6,1;W189,6,7,1;W185,7,7,1;W200,8,7,2;
G: W191,10,7,1;W187,18,7,1;W200,19,7,11;W186,30,7,1;
G: W187,6,8,1;W200,7,8,4;W188,11,8,1;W190,12,8,1;
G: W192,15,8,1;W184,17,8,1;W200,18,8,13,W188,31,8,1;
G: W195,2,9,1;W196,3,9,1;W184,5,9,1;W200,6,9,6;
G: W191,12,9,1;W184,14,9,1;W200,15,9,1;W188,16,9,1;
G: W185,17,9,1;W200,18,9,14;W189,1,10,1;W185,2,10,1;
G: W200,3,10,1;W188,4,10,1;W185,5,10,1;W200,6,10,7;
G: W188,13,10,1;W185,14,10,1;W200,15,10,17;W188,0,11,1;
G: W187,1,11,1;W200,2,11,94;W201,23,7,1;W201,10,8,1;
G: W202,23,8,1;W225,9,9,1;W224,10,9,1;W206,20,9,1;
G: W201,22,9,1;W203,23,9,1;W204,24,9,1;W201,7,10,1;
G: W207,8,10,1;W202,9,10,1;W201,19,10,1;W203,21,10,1;
G: W226,22,10,1;W205,24,10,1;W206,6,11,1;W202,7,11,1;
G: W202,13,11,1;W202,19,11,1;W201,24,11,1;W207,25,11,1;
G: W206,26,11,1;W201,5,12,1;W207,7,12,1;W203,13,12,1;W204,14,12,1;
G: W201,18,12,1;W225,23,12,1;W226,24,12,1;W203,27,12,1;
G: W204,28,12,1;W225,4,13,1;W224,5,13,1;W203,8,13,1;W204,9,13,1;
G: W205,14,13,1;W202,18,13,1;W227,23,13,1;W228,24,13,1;
G: W205,28,13,1;W208,0,14,430;W210,6,15,1;W222,7,15,1;
G: W213,8,15,1;W216,9,15,1;W217,10,15,2;W218,12,15,1;
G: W219,13,15,1;W220,14,15,1;W209,5,16,1;W211,6,16,1;
G: W232,7,16,1;W234,8,16,1;W149,9,16,5;
G: W221,14,16,1;W214,6,17,1;W149,7,17,2;W223,9,17,1;
G: W212,10,17,1;W215,7,18,1;W212,8,18,1;

G: MO,19
TH: $K$ $K$
G: W242,3,3,1;
G: W241,3,2,1;W240,2,3,1;W244,4,3,1;W243,3,4,1;

   R: Start sprites in motion.

G: S9,148,13,148,147,-50,0; S10,144,10,98,118,10,0;
G: S8,180,9,209,142,15,0; S2,132,3,172,158,0,0;
G: S6,128,3,161,151,0,0; S4,136,3,148,157,0,0;
G: S3,140,1,161,162,0,0;
G: S11,152,15,0,80,1,0; S12,156,15,112,67,1,0;
G: S13,160,15,190,57,1,0; S18,132,15,53,43,1,0;
G: S14,164,8,113,32,0,0;S15,168,8,126,32,0,0;
G: S16,172,8,113,48,0,0;S17,176,8,129,48,0,0;

G: MO,23;
TH: PRESS Q TO QUIT.
AS:
M: Q!q
JN: QA
```

```
    R: Stop all sprite handling.  If all sprites are not
    R: deleted they will remain active in the p-System
    R: and slow down the processing.  This may not be
    R: evident to the user.

 G: H2; H8; H4; H6; H9; H10; H11; H12; H13; H18;
 G: H3;H14;H15;H16;H17;T;F1;87;
 J;  MENU

    R:--- S O U N D ---

*SOUND PR: UGE
 T: $D9$ $D9$ $D9$
  : The sound list is being built.
 S: 1,C;V15,S99
 S: 1,N110,1;N110,1;N220,1;N220,1;N131,1;N131,1;N147,1;N147,1;N156,1;
 S: 1,N156,1;N165,1;N165,1;N131,1;N131,1;N165,1;N165,1;
 S: 1,N110,1;N110,1;N220,1;N220,1;N131,1;N131,1;N147,1;N147,1;N156,1;
 S: 1,N156,1;N165,1;N165,1;N131,1;N131,1;N165,1;N165,1;
 S: 1,N110,1;N110,1;N220,1;N220,1;N131,1;N131,1;N147,1;N147,1;N156,1;
 S: 1,N156,1;N165,1;N165,1;N131,1;N131,1;N165,1;N165,1;
 S: 2,C;V15;S99
 S: 2,N440,1;N220,1;N880,1;N440,1:N523,1;N262,1;N587,1;N294,1;N622,1;
 S: 2,N311,1;N659,1;N330,1;N523,1;N262,1;N659,1;N330,1;
 S: 2,N440,1;N220,1;N880,1;N440,1:N523,1;N262,1;N587,1;N294,1;N622,1;
 S: 2,N311,1;N659,1;N330,1;N523,1;N262,1;N659,1;N330,1;
 S: 2,N440,1;N220,1;N880,1;N440,1:N523,1;N262,1;N587,1;N294,1;N622,1;
 S: 2,N311,1;N659,1;N330,1;N523,1;N262,1;N659,1;N330,1;
 S: 1,P
 S: 2,P

 T: $C$ $D9$ $D9$ $D9$ $D9$ $D9$ $D9$
  : This is an excerpt from the jazzed-up
  : version of Bumble Bee Boogie.
 C: X=0

    R: wait for sound list to finish playing

*LOOP  C: X=X+1
 J(X<15): LOOP
 J: MENU
```

```
    R:--- S P E E C H ---

*SPEECH PR: UGE
 D: U$(40);U1$(40);U2$(40);U3$(40)
 C: U$="Hello, I am the TEXAS INSTRUMENTS"
 C: U1$="Hello, I am the #TEXAS INSTRUMENTS$"
 C: U2$="Home Computer.  I can say three hundred"
 C: U3$="seventy-three different words."
 T: $C$ $D9$ $D9$ $D9$ $D9$
  : $U$
 V: U1$
 T: $U2$
 V: U2$
 T: $U3$
 V: U3$
 J: MENU

    R:--- F I N I S ---

*END             R: This label ends the DEMO program.
 T: $C$
 E:
```

APPENDIX I. ERROR CODES

As you develop and run a TI PILOT program, you may make some mistakes that result in error messages. These error messages can come from several sources.

Error messages may be issued by the p-System as you use a specific utility program such as the Filer or Editor. Error messages may result from using the Editor as you compose the statements in a program. Then, too, error messages may result when you run a TI PILOT program.

If an error is found as a program is running, the TI PILOT interpreter displays a message identifying the kind of error and displays the statement containing the fault. The program continues with the next line.

The kind of error is identified by an error code and is reported as x-ERROR where x is one of the error codes in the list below. Following the error message, the statement containing the error is displayed and a caret (^) marks the place in the statement where the error was found.

Here is a list of the TI PILOT error codes.

A	Arithmetic error, such as attempting to divide by zero.
B	Subscript out of bounds.
C	Invalid Compute statement.
D	Disk error while reading a TI PILOT program.
E	Invalid expression.
F	File input or output error.
I	Illegal function argument.
J	Invalid jump destination.
L	Label space overflow.
M	Invalid modifier or conditioner.
O	Invalid op-code.
R	Out of memory space for arrays or strings.
S	Compute syntax error or source file format error.
T	Variable table overflow.
U	Wrong type of expression for context.
V	Invalid variable name or expression syntax error.
X	Execution error; more than one opcode on a line

THREE-MONTH LIMITED WARRANTY
HOME COMPUTER SOFTWARE MEDIA

Texas Instruments Incorporated extends this consumer warranty only to the original consumer purchaser.

WARRANTY COVERAGE

This warranty covers the case components of the software package. The components include all cassette tapes, diskettes, plastics, containers, and all other hardware contained in this software package ("the Hardware"). This limited warranty does not extend to the programs contained in the software media and in the accompanying book materials ("the Programs").

The Hardware is warranted against malfunction due to defective materials or construction. THIS WARRANTY IS VOID IF THE HARDWARE HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, IMPROPER SERVICE OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR WORKMANSHIP.

WARRANTY DURATION

The Hardware is warranted for a period of three months from the date of the original purchase by the consumer.

WARRANTY DISCLAIMERS

ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE THREE-MONTH PERIOD. TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR LOSS OF USE OF THE HARDWARE OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USER.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you in those states.

LEGAL REMEDIES

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

PERFORMANCE BY TI UNDER WARRANTY

During the above three-month warranty period, defective Hardware will be replaced when it is returned postage prepaid to a Texas Instruments Service Facility listed below. The replacement Hardware will be warranted for three months from date of replacement. Other than the postage requirement, no charge will be made for replacement.

TI strongly recommends that you insure the Hardware for value prior to mailing.

TEXAS INSTRUMENTS CONSUMER SERVICE FACILITIES

U.S. Residents:
Texas Instruments Service Facility
P.O. Box 2500
Lubbock, Texas 79408

Canadian Residents:
Geophysical Services Incorporated
41 Shelley Road
Richmond Hill, Ontario, Canada L4C5G4

Consumers in California and Oregon may contact the following Texas Instruments offices for additional assistance or information.

Texas Instruments Consumer Service
831 South Douglas Street
El Segundo, California 90245
(213) 973-1803

Texas Instruments Consumer Service
6700 Southwest 105th
Kristin Square, Suite 110
Beaverton, Oregon 97005
(503) 643-6758

IMPORTANT NOTICE OF DISCLAIMER REGARDING THE PROGRAMS

The following should be read and understood before purchasing and/or using the software media.

TI does not warrant that the Programs will be free from error or will meet the specific requirements of the consumer. The consumer assumes complete responsibility for any decision made or actions taken based on information obtained using the Programs. Any statements made concerning the utility of the Programs are not to be construed as express or implied warranties.

TEXAS INSTRUMENTS MAKES NO WARRANTY, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAMS AND MAKES ALL PROGRAMS AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL TEXAS INSTRUMENTS BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE PROGRAMS AND THE SOLE AND EXCLUSIVE LIABILITY OF TEXAS INSTRUMENTS, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE SOFTWARE MEDIA. MOREOVER, TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THE PROGRAMS.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you in those states.

Thanks to 99'er:

Ernie Pergrem

for the Scan of this Manual

Rework by:

TI99 Italian User Club in the year *2022*

(info@ti99iuc.it)

Texas Instruments invented the integrated circuit, the microprocessor, and the microcomputer. Being first is our tradition.

TEXAS INSTRUMENTS
INCORPORATED

Printed in U.S.A.