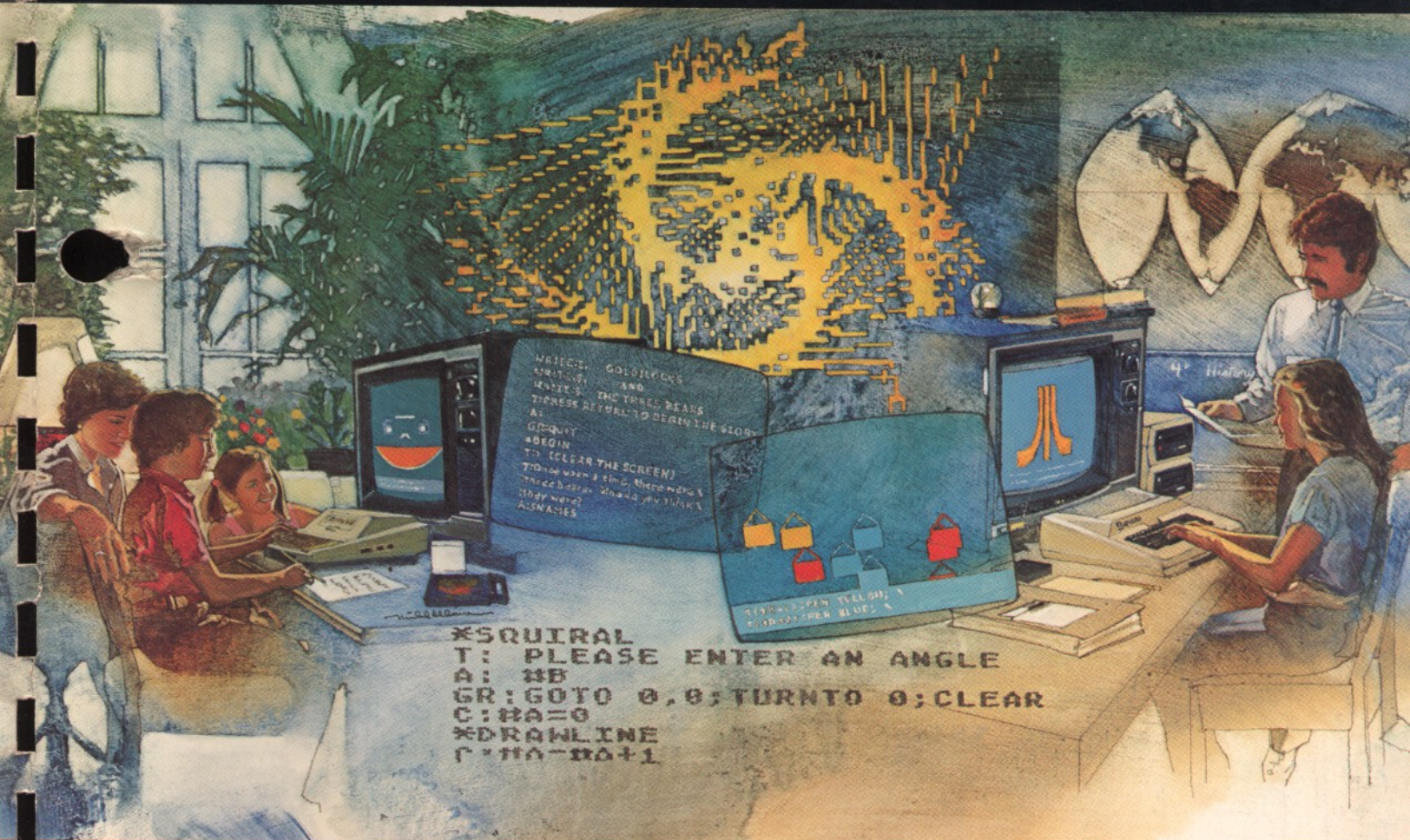


ATARI® 400/800™


PILOT PRIMER

THE PILOT PROGRAMMING LANGUAGE INSTRUCTION MANUAL



```
*SQUIRAL  
T: PLEASE ENTER AN ANGLE  
A: #B  
GR:GOTO 0,0;TURNTO 0;CLEAR  
C:HA=0  
*DRAWLINE  
P:HA-DA+1
```



A Warner Communications Company 


Use with
ATARI® 400™ or ATARI 800™
PERSONAL COMPUTER SYSTEMS

PILOT PRIMER

THE PILOT PROGRAMMING LANGUAGE

INSTRUCTION MANUAL



 A Warner Communications Company

ATARI wishes to acknowledge that the original manuscript for the *PILOT PRIMER* was written by Dorothy Kunkin and Keith Vann.

Every effort has been made to ensure that this manual accurately documents this product of the ATARI Computer Division. However, because of the ongoing improvement and update of the computer software and hardware, ATARI, INC. cannot guarantee the accuracy of printed material after the date of publication and cannot accept responsibility for errors or omissions.

Reproduction is forbidden without the specific written permission of ATARI, INC., Sunnyvale, CA 94086. No right to reproduce this document, nor the subject matter thereof, is granted unless by written agreement with, or written permission from the Corporation.

PRINTED IN U.S.A.

PROGRAM CONTENTS © 1980 ATARI, INC. MANUAL © 1980 DYMAX

ATARI PILOT owes a great deal to three outstanding computer educators:

Dr. John Starkweather (University of California Medical Center, San Francisco), who originated the first PILOT language for computer-based education.

Dr. Dean Brown, whose pioneering work at the Stanford Research Institute Education Laboratory between 1967 and 1974 showed PILOT to be a language for children's programs, as well as for teachers.

Dr. Seymour Papert, at the LOGO Project at Massachusetts Institute of Technology, for the development of "Turtle Geometry" and "Turtle Graphics," which ATARI PILOT uses.

PREFACE

WHAT IS PILOT?

PILOT stands for **Programmed Inquiry, Learning Or Teaching**. PILOT is a simple, flexible language that is easy to learn in a short period of time. It is an excellent tool to introduce children and beginners to computers and computer programming. PILOT was created as a text-oriented language for beginners who do not have an extensive mathematical background. With the basic knowledge of computers that can be acquired through PILOT's simple programming constructs, PILOT can provide a solid base for future computer learning.

PURPOSE OF THIS MANUAL

This manual is designed to:

- Teach you how to use the PILOT language on the ATARI® Personal Computer System
- Provide an introduction to simple graphics and sound through the medium of **ATARI PILOT**

Our goal is to enable you to design educational programs that respond appropriately to a variety of inputs. This means tutorials, programs using interactive dialogues, and games that genuinely respond to you. We want you to learn PILOT and be able to use it successfully to meet your own goals.

If you are a teacher or parent, this manual will enable you to design interactive curriculum materials that provide personalized, self-paced instruction for your students or children.

HOW TO USE THIS MANUAL

This manual is specifically designed for the beginner who may know little or nothing about computer programming. It begins with simple concepts which grow as you proceed through the book. Sections 1 through 10 provide the fundamentals of ATARI PILOT; therefore, all beginning readers should start with the Introduction and proceed sequentially through Section 10. Readers who are advanced programmers or are familiar with the PILOT language may want to read the Introduction and each Section Summary before skipping ahead to the advanced sections.

Each section teaches a new concept. The sections begin with a checklist of the major topics that will be covered in Concepts Introduced. At the end of each section is a Summary that briefly reviews the contents of the section, followed by a Quiz to test your understanding of the new concepts. Following the Quiz is a section explaining the more difficult concepts for the advanced programmer.

CAST OF COLORS

The four colors below will appear throughout this manual to highlight special words of advice. Each color is related to a specific kind of message.

- Provides helpful hints on how to do things. Often offers little reminders. ■
- Usually introduces a new topic or explains a difficult concept. ■
- Lets you know about things you should do to avoid problems. ■
- Will appear throughout the manual, to explain the meaning of the special symbols we use such as quotes (""), brackets ([]), and others. ■

CONTENTS

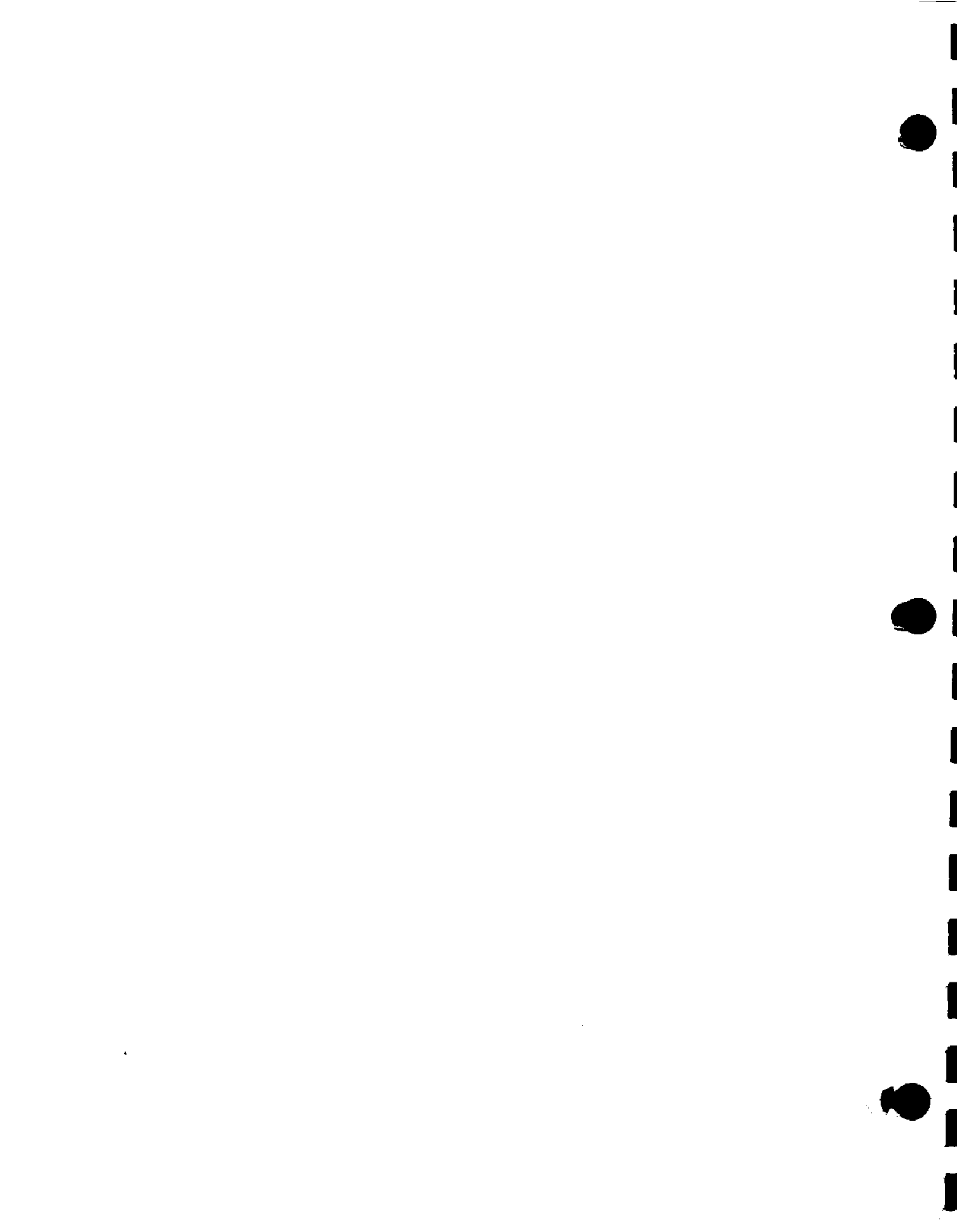
PREFACE	iii
What Is PILOT?	iii
Purpose of This Manual	iii
How To Use This Manual	iii
Cast of Colors	iv
<hr/>	
1 INTRODUCTION	1
Getting To Know Your Computer	1
Getting To Know the Keyboard	2
<hr/>	
2 GETTING STARTED WITH PILOT	5
Concepts Introduced	5
What Is the ATARI PILOT Cartridge?	5
Changing the Screen Display	5
Clearing the Screen	6
Correcting Mistakes	7
Some PILOT Commands	8
What Is a Command?	8
The TYPE Command, T:	9
The SOUND Command, SO:	9
The GRAPHICS Command, GR:	13
Section 2 Summary	16
Section 2 Quiz	17
<hr/>	
3 WRITING A PILOT PROGRAM	19
Concepts Introduced	19
Immediate and Deferred Commands	19
What Is a Program?	19
Executive Commands	20
The RUN Command	20
The LIST Command	21
The NEW Command	23
The AUTO Command	23

More Commands	26
The TYPE Command, T:	26
The ACCEPT Command, A:	26
The REMARK Command, R:	31
The END Command, E:	32
Correcting and Changing Your Programs	32
Erase a Line	32
Insert a Line	33
Replace a Line	34
Line Renumbering	35
The RENUMBER Command, REN	35
Fancy Renumbering With REN	36
Section 3 Summary	37
Section 3 Quiz	38
<hr/>	
4 DECISION-MAKING PROGRAMS	41
<hr/>	
Concepts Introduced	41
Matching for Clues	42
The Function of the ACCEPT Command, A:	42
The MATCH Command, M:	42
Conditional Commands	44
The TYPE IF YES and TYPE IF NO Commands, TY:, TN:	44
Putting It All Together	47
The END IF YES and END IF NO Commands, EY:, EN:	48
Section 4 Summary	50
Section 4 Quiz	51
Advanced Decision-Making Techniques	53
Separating MATCH Strings With Vertical Bars	53
<hr/>	
5 BRANCHING	55
<hr/>	
Concepts Introduced	55
The Statement Label	55
The JUMP Command, J:	55
The JUMP IF YES and JUMP IF NO Commands, JY:, JN:	57
The TRACE Command	59
Section 5 Summary	61
Section 5 Quiz	62
Advanced Branching Techniques	63
The JUMP Command in Immediate Mode, J:	63
The JUMP ON MATCH Command, JM:	63
Mini-Quiz	65

6	VARIABLES	67
	Concepts Introduced	67
	What Is a Variable?	67
	String Variables	69
	What Is a String Variable?	69
	Defining String Variables	69
	Using TYPE and ACCEPT Commands	
	With String Variables	70
	What Are the Values of Your String Variables?	72
	The DUMP Command	72
	The COMPUTE Command, C:	72
	String Concatenation	73
	Growing Strings	73
	Generating Plural Strings	75
	The ACCEPT Command Without Input	76
	The COMPUTE IF YES	
	and COMPUTE IF NO Commands, CY:, CN:	77
	Clearing Your Variables	77
	Section 6 Summary	78
	Section 6 Quiz	79
	Advanced Programming With Variables	81
	Assigning Variables Into Other Variables	81
	String Indirection	81
	The MATCH STRING Command, MS:	81
	Sensing a Match	82
	Mini-Quiz	83
7	USING NUMBERS	85
	Concepts Introduced	85
	Maximum Number Size	85
	Modulo	85
	Numeric Types	86
	Constants	86
	Numeric Variables	87
	Assigning Value to Numeric Variables	87
	The COMPUTE Command, C:	87
	The Random Number	89
	Arithmetic Operators	89
	Operator Precedence	89
	Relational Operators	89
	Section 7 Summary	91
	Section 7 Quiz	93

8	MODULES	95
	Concepts Introduced	95
	What Is a Module?	95
	The Parts of a Module	96
	Calling a Module	96
	The USE Command, U:	96
	The Module End	97
	The END Command, E:	97
	Nested Modules	97
	Conditional Modules	98
	The USE IF YES and USE IF NO Commands, UY:, UN:	98
	Example: The Calculator Program	99
	Section 8 Summary	101
	Section 8 Quiz	102
9	MANY VOICES, MANY SOUNDS	105
	Concepts Introduced	105
	Definition List	105
	The SOUND Command, SO:	106
	Immediate Mode Sounds	107
	Program Mode Sounds	108
	The PAUSE Command, PA:	108
	Random Music	109
	A Special Song	110
	Section 9 Summary	111
	Section 9 Quiz	112
	Advanced Programming With Sound	114
	The PAUSE Command: Increments of 1/60ths	114
10	WELCOME TO GRAPHICS	115
	Concepts Introduced	115
	The GRAPHICS Command, GR:	116
	The Graphics Screen	117
	Cartesian Coordinates	117
	Turtle Geometry	118
	GRAPHICS Subcommands	120
	The PEN Subcommand	120
	Subcommands Using Cartesian Coordinates	121
	The GOTO Subcommand	121
	The DRAWTO Subcommand	123
	The FILLTO Subcommand	124

Subcommands Using Turtle Geometry	125
The TURNTO Subcommand	125
The TURN Subcommand	126
The DRAW Subcommand	128
The GO Subcommand	129
The FILL Subcommand	130
Repeating GRAPHICS Commands	131
Home, Sweet Home Program	133
Section 10 Summary	134
Section 10 Quiz	137
Advanced Programming With GRAPHICS	139
GRAPHICS Variables	139
%X	139
%Y	139
%A	139
%Z	139
Mini-Quiz	141



APPENDICES

A	CONNECTING YOUR ATARI COMPUTER SYSTEM	143
	Concepts Introduced	143
	Connecting Your ATARI Computer System	143
	Inserting PILOT	147
	Turning On PILOT in Your ATARI Computer Without a Disk Drive	147
	Turning On PILOT in Your ATARI Computer With an ATARI 810 Disk Drive	147
	Turning Off Your ATARI Computer System	148
B	SPEAKING TO DEVICES	149
	Concepts Introduced	149
	The ATARI 810 Disk Drive	150
	Speaking to Your ATARI 810 Disk Drive	151
	Saving a Program on a Diskette	151
	Write-Protecting a Diskette	152
	Loading a Program From a Diskette	152
	Merging Programs	153
	The DOS Utility	155
	The Disk Directory	156
	Using the DOS Disk Directory	156
	Creating Your Own Disk Directory	156
	Returning to PILOT From the DOS Utility	158
	DON'T PANIC! (How to Handle Trouble With the Disk Drive)	158
	Screen Displays "Boot Error"	158
	BUSY Light on Disk Drive Does Not Go Off	159
	The Computer Does Not Respond	159
	The ATARI 410 Program Recorder	159
	Speaking to Your Program Recorder	159
	Saving a Program on a Cassette Tape	159
	Write-Protecting a Cassette Tape	161
	Loading a Program From a Cassette Tape	161
	Merging Programs	162
	A Method to Manage Programs on Cassette Tapes	162
	Saving the Cassette Directory	162
	Adding Programs to the Directory and the Tape	165

The ATARI Printers	165
Printing a Program	166
<hr/>	
C SCREEN EDITING	167
<hr/>	
Concepts Introduced	167
The Screen Editor	167
The Screen Editing Keys	168
The RETURN Key	168
The SHIFT Key	168
The CAPS/LOWER Key	168
The TAB Key	168
TAB Set	168
TAB Clear	169
TAB	169
The CTRL Key	169
The Inverse Video Key	169
The CLEAR Key	169
The Cursor Control Keys	170
Cursor Up	171
Cursor Down	172
Cursor Right	172
Cursor Left	174
Editing the Current Display Line	175
Backspacing	175
Deleting	177
Character Deletion	177
Line Deletion	178
Inserting	180
Character Insertion	180
Line Insertion	181
Deferred Cursor Movements	182
<hr/>	
D CONTROLLERS	185
<hr/>	
Concepts Introduced	185
Sensing	185
Sensing the Position of the Joystick	185
Sensing the Joystick's Trigger	187
Sensing the Position of the Paddle	188
Sensing the Paddle Trigger	189
<hr/>	
E PILOT I/O ERROR CODES	191
<hr/>	

ILLUSTRATIONS

1-1	ATARI Personal Computer System	1
1-2	The ATARI Keyboard	2
1-3	The ATARI Keyboard: Graphics Keys	2
1-4	The ATARI Keyboard: Cursor Control Keys	3
2-1	Clearing the Screen	7
2-2	Piano Keyboard	10
2-3	Whole and Half Steps	10
2-4	C Major Scale	11
2-5	C Major Triad	12
2-6	C Major Chord	13
2-7	GR: DRAW 10	13
2-8	Clearing the GRAPHICS Screen	14
2-9	Drawing a Box	14
2-10	Fill With Yellow	15
2-11	Return to Text Mode From Graphics Mode	15
3-1	Program Run	21
3-2	AUTO Mode	24
3-3	Return to Text Mode From AUTO Mode	25
3-4	Program List and Run	26
3-5	NAME Program Run	28
3-6	CLOTHES Program Run	29
3-7	TYPE and ACCEPT Commands: Program Run	30
4-1	Concept of the Accept Buffer	42
4-2	MATCH Command: Sample Runs	46
4-3	Sample Runs of the TO GO OR NOT TO GO Program	48
6-1	Concept of Variables	68
6-2	Assigning Variable Names	68
6-3	Assigning Values to Variables	68
6-4	Changing a Variable's Value	69
6-5	Defining a Variable With Input	70
6-6	String Variables: Program Run	70
6-7	Growing Strings: Program List and Run	74
6-8	Growing Strings With a Second DUMP Command	75
6-9	ACCEPT Without Input: Program Run	77

7-1	Division Problem	86
7-2	The Modulo of 7\3	86
7-3	FOREVER Program Run	91
8-1	Program With Several Modules	95
8-2	Module Structure	96
8-3	Nested Modules	98
9-1	Position of Middle C	105
9-2	Relation of Piano Keyboard to Musical Scale	106
9-3	Musical Scales	107
9-4	PAUSE Values for Moderato (Andante) Tempo	108
9-5	PAUSE Values for Different Tempos	109
9-6	PAUSE Values in 60ths of a Second	114
10-1	The GRAPHICS Screen	116
10-2	Cartesian Coordinates	117
10-3	The GRAPHICS Turtle	118
10-4	The 0- and 360-Degree Angle	119
10-5	Calculating Directional Angles	119
10-6	GR: DRAWTO 20,20	121
10-7	The Four Quadrants of the GRAPHICS Screen	122
10-8	Filling the Screen With Dots Using GOTO	123
10-9	Drawing a Box	124
10-10	Filling With FILLTO	125
10-11	Turning the Turtle Toward 90 Degrees	126
10-12	Turning the Turtle Toward - 45 Degrees	126
10-13	Turning the Turtle 45 Degrees to the Right	127
10-14	Turning the Turtle 60 Degrees to the Left	127
10-15	TURN Subcommand: Program Run	128
10-16	DRAW Subcommand: Program Run	129
10-17	GO Subcommand: Program Run	130
10-18	Using the FILL Subcommand	131
10-19	Drawing a Box With Repeating GRAPHICS Commands	132
10-20	Drawing a Star With Repeating GRAPHICS Commands	132
10-21	Home, Sweet Home Program Run	133
A-1	The ATARI 810 Disk Drive	143
A-2	The ATARI 410 Program Recorder	143
A-3	The ATARI 820, 822, and 825 Printers	143
A-4	The ATARI 400/800 Computer With Program Recorder	144
A-5	The ATARI 400/800 Computer With Disk Drive	145
A-6	The ATARI 400/800 Computer With Program Recorder and Printer	145
A-7	The ATARI 400/800 Computer With Disk Drive and Printer	146
A-8	The ATARI 400/800 Computer With Disk Drive, Printer, and Program Recorder	146

B-1	Cassette Tape	150
B-2	Diskette	150
B-3	DOS Utility Program	155
B-4	Sample Directory	156
C-1	The ATARI Keyboard: Screen Editing Keys	168
C-2	Using the Inverse Video Key	169
C-3	The ATARI Keyboard: The Cursor Control Keys	170
C-4	Moving the Cursor Over a Character	170
C-5	Using the Cursor Up Key	171
C-6	Cursor Up Wraparound	171
C-7	Using the Cursor Down Key	172
C-8	Cursor Down Wraparound	172
C-9	Using the Cursor Right Key	173
C-10	Cursor Right Wraparound	173
C-11	Using the Cursor Left Key	174
C-12	Cursor Left Wraparound	174
C-13	Backspacing the Cursor Using the DELETE BACK S Key	177
C-14	Single-Character Deletion	177
C-15	Multiple-Character Deletion	178
C-16	Character Deletion Completed	178
C-17	Line Deletion: Positioning the Cursor	179
C-18	Line Deletion Completed	179
C-19	Character Insertion: Positioning the Cursor	180
C-20	Character Insertion: Creating a Space	180
C-21	Character Insertion Completed	181
C-22	Line Insertion: Positioning the Cursor	181
C-23	Line Insertion: Creating a Space	182
C-24	Line Insertion Completed	182
C-25	Character Spiral Program Run	184
D-1	Joystick Positions	185
D-2	Sample Joystick Sensing	186
D-3	Sample Joystick Trigger Sensing	187
D-4	Paddle Positions	188
D-5	Sample Paddle Positions	188



INTRODUCTION

GETTING TO KNOW YOUR COMPUTER

WHAT IS A COMPUTER, ANYWAY?

A computer is a tool that can do many special things for you. A computer doesn't have a personality and it doesn't think; it can only do what it is told to do.

Basically, a **computer is a machine with a memory** with the capability of making comparisons. You can tell it to remember or **store** information in its memory. Afterwards, you can ask it to tell you what it has stored in its memory. The computer can perform calculations, sort the information stored in its memory, and engage in a dialogue with you, all at lightning speed.

When you sit down in front of your computer, you become closely associated with the **system**. Here is what the system looks like:

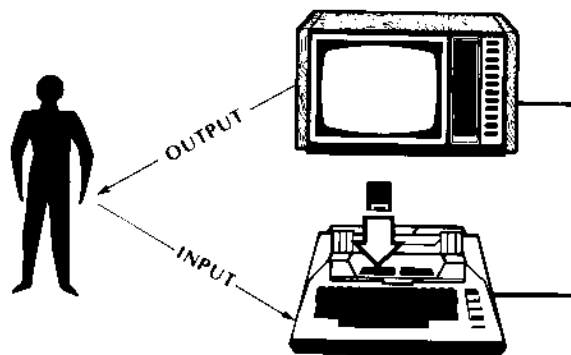


Figure 1-1 ATARI® Personal Computer System

What you say to the computer is called **input**. The computer's response is called **output**. Giving instructions to the computer is called **computer programming**. The art of computer programming is to make sure that:

- The computer understands your input
- The output makes sense to you

Above all, remember one thing: it is **you** who makes the system work. Without you to tell it what to do, the computer is just another machine.

GETTING TO KNOW THE KEYBOARD

Here is the ATARI computer keyboard:



Figure 1-2 The ATARI Keyboard

The keyboard is similar to a typewriter. To type the character on the top half of the key, you must press and hold the **SHIFT** key while you press the key. To type the character on the bottom of the key, simply press the key.

The keyboard is made up of five types of characters: alphabetic, numeric, punctuation, graphics, and cursor control. The alphabetic characters (letters A through Z), numbers (0 through 9), and punctuation characters are clearly marked on the key caps. To type lowercase (small) letters, press the **CAPS LOWER** key first to put the computer in lowercase mode. To change to uppercase letters, press the **CAPS LOWER** key and the **SHIFT** key.

The graphics characters are not labeled on the key caps but are accessible from the alphabetic keys, as shown below:

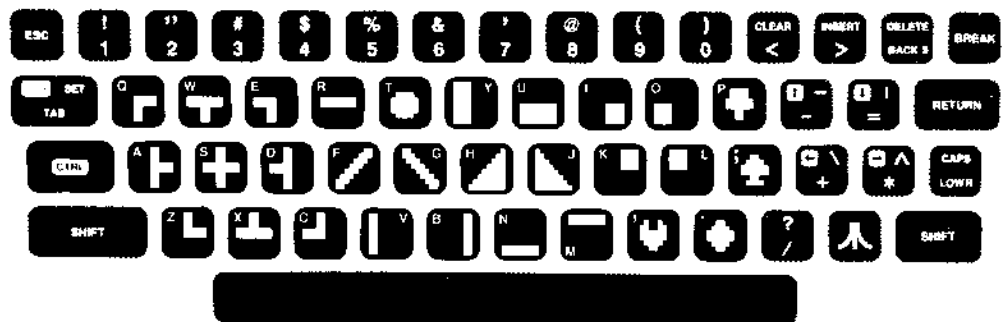


Figure 1-3 The ATARI Keyboard: Graphics Keys

To access the graphics characters you must hold down the **CTRL** key (left side) and then press the appropriate graphics keys.

The cursor control keys move the cursor in any direction to change, or edit, the screen display. **The cursor is the square that marks your place on the screen.** The cursor control keys are shown below:

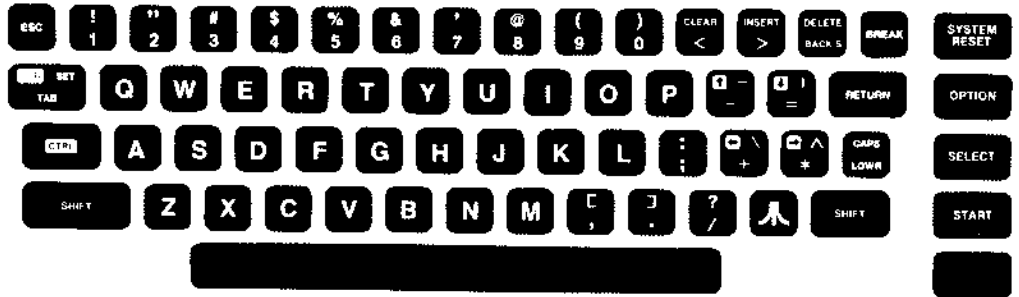


Figure 1-4 The ATARI Keyboard: Cursor Control Keys

This keyboard has a special feature: **repeating keys**. If you hold down a key for one second or longer, it will continue typing until you release the key. Each time a character is typed repeatedly, the computer makes a beeping noise. Repeating keys is a convenient feature, but be careful; it's fast!



GETTING STARTED WITH PILOT

CONCEPTS INTRODUCED

What Is the ATARI PILOT Cartridge?
 Changing the Screen Display
 Clearing the Screen
 Correcting Mistakes
 Some PILOT Commands
 What Is a Command?
 The TYPE Command, T:
 The SOUND Command, SO:
 The GRAPHICS Command, GR:

Before getting started with the PILOT language you must turn on your system **with your ATARI PILOT cartridge inserted**. Instructions on how to insert the cartridge can be found in Appendix A, Inserting PILOT.

WHAT IS THE ATARI PILOT CARTRIDGE?

The ATARI PILOT cartridge is a small boxlike unit containing special instructions for your ATARI computer about the PILOT language. When the ATARI PILOT cartridge is inserted into your computer it becomes part of the system, allowing you to communicate with your computer in the PILOT language. If the cartridge is not inserted into your computer, the computer cannot understand the ATARI PILOT language, and is useful only as a memo-pad.

Once the cartridge is inserted, your screen should look like this:

```
ATARI PILOT (C) COPYRIGHT ATARI 1980
```

```
READY
█
```

The cursor should be positioned at the beginning of the line below R in READY. The cursor marks your place.

CHANGING THE SCREEN DISPLAY

Type: **HELLO**. As you type **HELLO** the cursor moves one position to the right, marking the place of the next character to be typed. The cursor should now be positioned after the O, waiting for further input.

```
ATARI PILOT (C) COPYRIGHT ATARI 1980
```

```
READY
HELLO█
```

Press the **RETURN** key. The computer responds immediately to your input as you press the **RETURN** key.

```
ATARI PILOT (C) COPYRIGHT ATARI 1980
```

```
READY
HELLO
```

```
HELLO
***WHAT'S THAT?***
█
```

The computer responded with a ***** WHAT'S THAT? ***** message because it didn't understand your input. When the computer doesn't understand your input, it displays your input followed by a ***** WHAT'S THAT? ***** message.

■ As a rule, you must always press the **RETURN** key at the end of a line. Pressing the **RETURN** key tells the computer that you have finished typing and that it's the computer's turn to respond. ■

Let's input something the computer understands. Type: **T: HELLO**. (Don't press the **RETURN** key yet!)

■ To type a colon(:) hold down the **SHIFT** key and then press the **;/** key. ■

The computer understands commands. The **T:** is a **TYPE** command. This particular command tells the computer to type **HELLO**. Press the **RETURN** key for the computer's response. The screen should display:

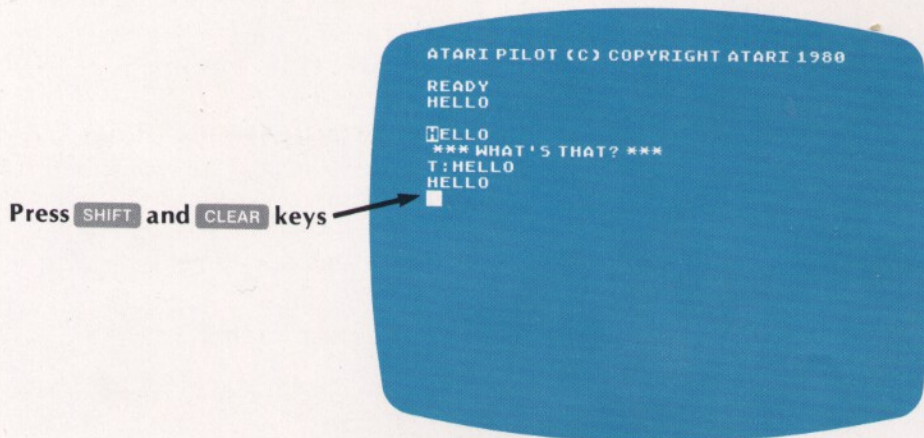
```
ATARI PILOT (C) COPYRIGHT ATARI 1980
READY
HELLO
HELLO
*** WHAT 'S THAT? ***
T:HELLO
HELLO
■
```

Because the computer understands the command it responds correctly by typing **HELLO**. Commands will be discussed at the end of this section and in the following two sections.

CLEARING THE SCREEN

Before proceeding, let's erase, or *clear* the screen. Clearing the screen is simple; hold down the **SHIFT** key, and at the same time press the **CLEAR** key (top row).

Clear the screen. POOF! Look what happens!



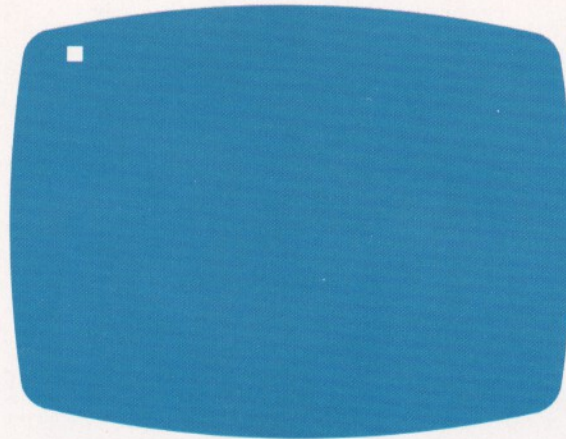


Figure 2-1 Clearing the Screen

Your screen is now a blank screen except for the cursor waiting in the upper left corner for more input.

■ Always start each new activity by clearing the screen.

NEVER press the `SYSTEM RESET` key to clear the screen. ■

CORRECTING MISTAKES

There are three methods to correct spelling or typing mistakes when talking to your computer on the keyboard. They are:

1. Retype the entire line.
2. Backspace to the mistake and retype the rest of the line.
3. Use cursor control keys to correct the mistake. This method is for advanced programmers and is discussed in Appendix C.

1. Retype the Entire Line

Type: **T: THIS IS MY FURST TIME** (Do not press the `RETURN` key.) Does your screen look like this?:

```
T: THIS IS MY FURST TIME■
```

OOPS! We misspelled FIRST as FURST! To correct this mistake you must first move the cursor to a blank line by pressing the `RETURN` key. Because this is a TYPE command, when you press the `RETURN` key the cursor first drops down to the following line to display the message before waiting on a blank line below:

```
T: THIS IS MY FURST TIME  
THIS IS MY FURST TIME  
■
```

Now, retype the entire line, spelling FIRST correctly. When it is typed, press the `RETURN` key.

```
T: THIS IS MY FURST TIME
THIS IS MY FURST TIME
T: THIS IS MY FIRST TIME
THIS IS MY FIRST TIME
■
```

Your line is now correct.

■ To create more space between input and output lines press the **RETURN** key to drop the cursor down one line. Or, if the screen below the cursor is blank, you can easily move the cursor down the left side of the screen by pressing the **RETURN** key several times. ■

2. Backspace to the Mistake and Retype the Rest of the Line

To begin this activity, clear the screen by pressing the **CLEAR** key, with the **SHIFT** key. Type: **T: THIS IS MY FURST TIME** (Do not press the **RETURN** key yet.)

```
T: THIS IS MY FURST TIME■
```

This following exercise erases all the letters to the right of the F (URST TIME) and retypes them correctly.

To *backspace* the cursor **and** erase the letters, press the **DELETE BACK S** key (top row) for each letter to be erased. Each time the **DELETE BACK S** key is pressed, the cursor will move over and erase the character to the immediate left. If we press the **DELETE BACK S** key nine times, or hold the key down for nine repeating key movements, the screen will change from this:

```
T: THIS IS MY FURST TIME■
```

to this:

```
T: THIS IS MY F■
```

Once the cursor has erased the mistake, retype the rest of the line correctly.

```
T: THIS IS MY FIRST TIME■
```

If you've made no further mistakes, press the **RETURN** key. Look at your screen. Your sentence is perfect! (And no one will ever know you made a mistake!)

```
T: THIS IS MY FIRST TIME
THIS IS MY FIRST TIME
■
```

SOME PILOT COMMANDS

WHAT IS A COMMAND?

A **command** is an instruction to the computer to carry out a specified task. In PILOT there are about ten principal commands that instruct the computer to respond in different ways. Each command is made up of two parts: the *command label* and the *action of the instruction*. The command label is usually one to two letters in length, followed by a colon (:).

THE TYPE COMMAND, T:

The TYPE command lets you type messages on the screen. The format for the TYPE command is:

T: *message*

Whatever you enter following the colon will be your *message*. Here are some examples:

T: HELLO
T: MY NAME IS GODZILLA
T: BYE!

■ Remember, to type a colon (:), hold down the **SHIFT** key and press the **;/** key at the same time. ■

Clear the screen and type in: **T: GREETINGS!** and press **RETURN**. Your screen should look like:

```
T: GREETINGS!  
GREETINGS!  
■
```

The computer typed your message back! This is another example of an interaction between you and the computer; you tell it to type a message and it responds by typing the message. Whatever you enter following the **T:** the computer displays on the following line. If nothing is entered following the **T:** the computer prints a blank line. Again, the cursor waits on the line below the computer's message.

Ready for another one?

Type: **T: MY NAME IS BARRY** and press **RETURN** (enter your own name instead of Barry).

Does your screen display this message:

```
T: HELLO  
HELLO  
T: MY NAME IS BARRY  
MY NAME IS BARRY  
■
```

Putting information on the screen is as easy as can be. Go ahead and make up some of your own!

THE SOUND COMMAND, SO:

Let's try something different. Did you know that your computer is a musical instrument? The SOUND command lets you create different musical tones (or notes) that you hear through your television speaker. So, turn up the volume on your television set.

Each musical tone within the computer's range has been assigned a numerical value from 0 to 31. The range of the computer contains all the tones between the C below middle C and the F# (F sharp) 18 tones above middle C. On the piano keyboard shown below you can see the keys, their names, and their numeric values.

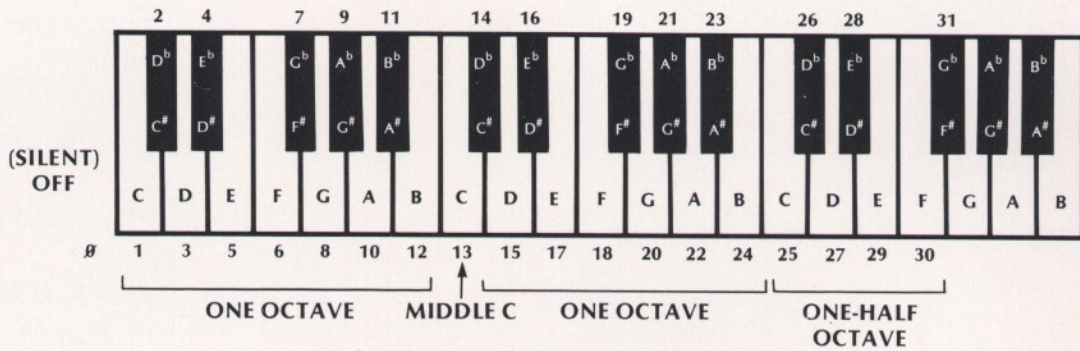


Figure 2-2 Piano Keyboard

Examine the keyboard illustration for a minute. You can see that it is made up of white keys and black keys. Each time you move from one key to another, you are moving an *interval*. The simplest interval is a half step. To play a half step, you move from one key to another without leaving any keys between them. C to C#, E to F, and Bb to B are each half-step intervals. A whole-step interval is almost as simple. A whole step is two half steps. To play a whole step, move from Eb to F or C# to D#; these are both whole-step intervals.

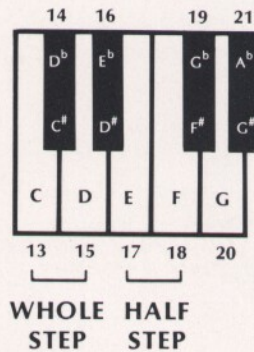


Figure 2-3 Whole and Half Steps

If you were to play the tones from middle C to the C above middle C on a piano using only half-step intervals, you would be playing a *chromatic scale*. Using the SOUND command and the numeric values for each of the keys, you can "play" a chromatic scale using your computer.

Clear the screen. Type **SO: 13** **RETURN**.

Hear that? That is middle C. To play the next tone in the chromatic scale, type **SO: 14** **RETURN**. Did you hear the tone change? Now, type in the following SO: commands to complete your chromatic scale. Be sure to press **RETURN** after you enter each numeric value.

SO: 15
SO: 16
SO: 17
SO: 18
SO: 19
SO: 20
SO: 21
SO: 22
SO: 23
SO: 24
SO: 25

■ Press the **BREAK** key to stop the sound. The **BREAK** key is located in the upper right corner of your keyboard. ■

If you would like to, you can play a chromatic scale beginning on D by typing **SO: 14** for your first command, continuing to enter the succeeding commands and numeric values until you reach **SO: 26**. In fact, you can play a chromatic scale starting with any numeric value between 1 and 19.

A *major scale* is slightly different from a chromatic scale. You only use 8 numeric values instead of the 13 you used for the chromatic. For example, the C major scale consists of the tones C, D, E, F, G, A, B, and C. This type of scale uses both whole-step and half-step intervals. The illustration below shows the format of a major scale using the C scale as an example.

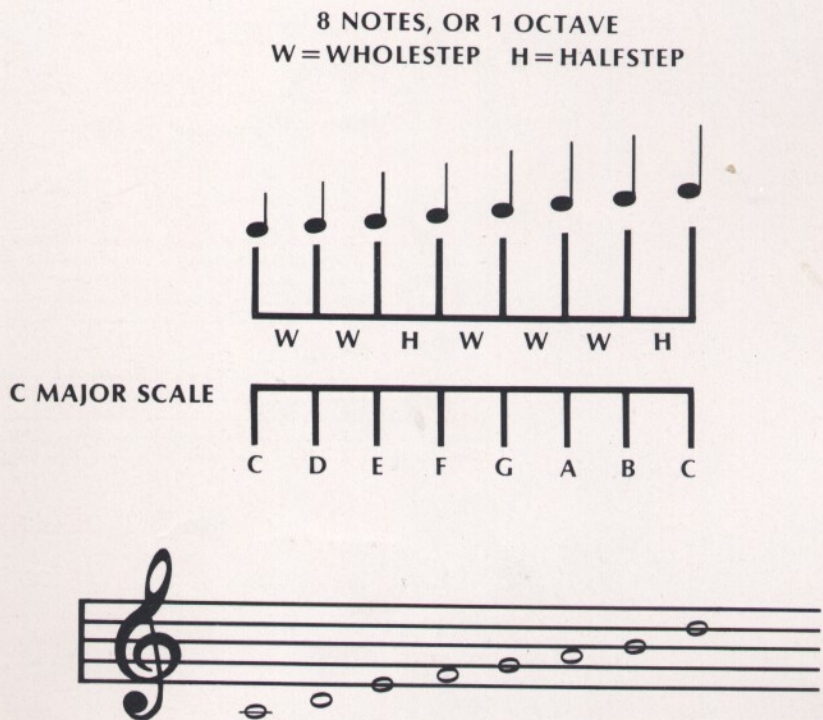


Figure 2-4 C Major Scale

This format works with any major scale whether you begin on C or F#. But, for the moment, let's stick to the C major scale. To "play" a C major scale using your computer, type in the following SOUND commands with their attached numeric values. Don't forget to press the **RETURN** key after each line:

SO: 13
SO: 15
SO: 17
SO: 18
SO: 20
SO: 22
SO: 24
SO: 25

Press **BREAK** to stop the sound. Or, you can type **SO: 0** **RETURN** to stop the sound. A value of 0 does not have a tone associated with it.

Up to now, you have been using only one tone for each SOUND command. However, you can use the SOUND command to play more than one tone. On a piece of sheet music, you have different notes for soprano, alto, tenor, and bass voices. Using the SOUND command and the numeric values of notes, you can create your own voices and harmony. The format for using multiple voices with one SOUND command is:

SO: voice1, voice2, voice3, voice4

Now you can play chords that consist of three or more tones.

Type:

SO: 13,17,20 and press **RETURN**
SO: 0 and press **RETURN**

This is a simple C major chord.

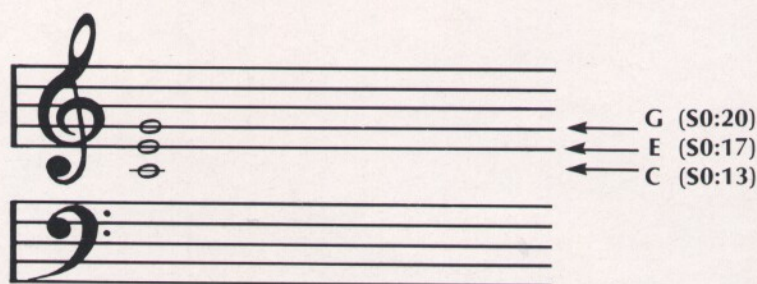


Figure 2-5 C Major Triad

Now clear the screen and type:

SO: 13,17,20,25 and press **RETURN**
SO: 0 and press **RETURN**

■ When using multiple voices, each voice must be separated by a comma or a space. ■

You have now added an extra C to your C major chord.

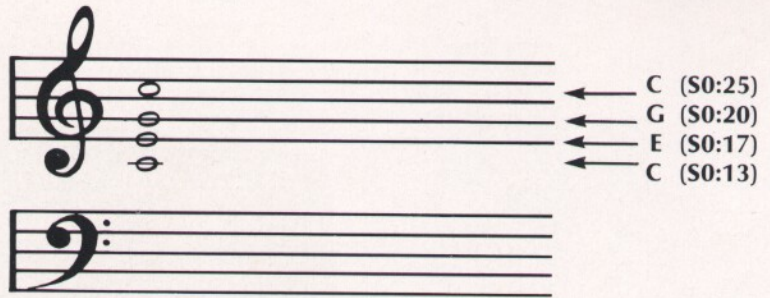


Figure 2-6 C Major Chord

Go ahead and experiment with creating your own single and multiple voices by inserting any combination of voices between 1 and 31. For further information on the SOUND command, refer to Section 9, Many Sounds, Many Voices.

THE GRAPHICS COMMAND, GR:

The GRAPHICS command, tells the computer to enter the *graphics mode and open the graphics screen*. Once in graphics mode, you can draw points and lines, or fill spaces in three colors: red, blue, and yellow.

Clear the screen and enter the following command: **GR: DRAW 10**. Press the **RETURN** key and watch what happens:

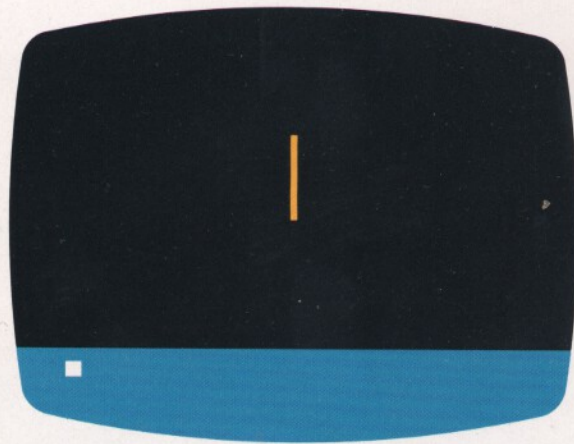


Figure 2-7 GR: DRAW 10

Did the screen turn black except for a blue strip at the bottom? Did a yellow (or white) line appear in the middle of the screen? (Both answers should be yes.)

The GR: command tells the computer to enter graphics mode by displaying a black screen. The blue strip at the bottom allows you to display up to four lines of text without destroying the graphics.

The vertical line appeared on the screen when the DRAW 10 told the computer to draw that line. Section 10, Welcome to Graphics, explains exactly how to draw that line or any other figure in graphics mode.

Let's try something using lost of color. To clear the graphics screen and remain in graphics mode, type: **GR: CLEAR** and press **RETURN**.



Figure 2-8 Clearing the GRAPHICS Screen

Here's how easy it is to draw a box. Type this:

GR: 4(DRAW 20; TURN 90) **RETURN**

■ The semicolon does not require the **SHIFT** key. ■

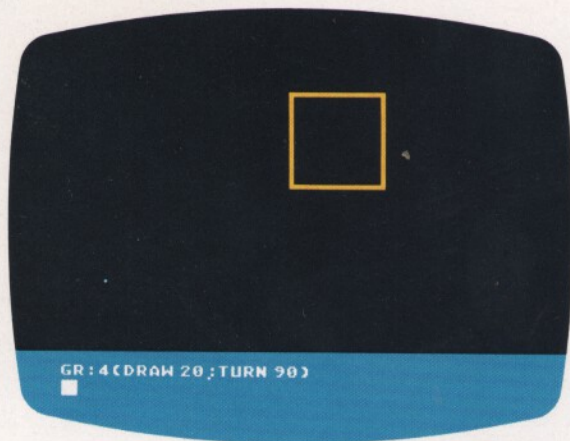


Figure 2-9 Drawing a Box

Now, type: **GR: FILL 40** and press **RETURN**. Look at the blue strip at the bottom to see what you type in. Press the **RETURN** key and watch the box and the top half of your screen fill up with **YELLOW**:



Figure 2-10 Fill With Yellow

To return to *text mode*, type: **GR: QUIT** and press **RETURN**. WOOSH! Now you're back to text mode with the blue screen and the cursor waiting at the upper left corner.



Figure 2-11 Return to Text Mode From Graphics Mode

SECTION 2 SUMMARY

ATARI PILOT uses a system called **TURTLE GRAPHICS** to draw the preceding graphics. Turtle Graphics can be explored further in Section 10, Welcome to Graphics.

If your ATARI computer is turned on correctly with the ATARI PILOT cartridge plugged in, the screen should display:

```
ATARI PILOT (C) COPYRIGHT ATARI 1980
```

```
READY
```



If your screen doesn't look like this, then follow the start-up procedure for your system in Appendix A.

The *cursor* marks your place on the screen. The cursor moves as you enter input.

The READY message means that the computer is finished *responding* and is waiting for further input.

To end a line of input, press the **RETURN** key.

To clear the screen, hold down the **SHIFT** key and press the **CLEAR** key.

To correct typing or spelling mistakes you can either retype the entire line, or backspace the cursor to the mistake with the **DELETE BACK S** key and retype the line from there.

A *command* is an instruction to the computer to carry out a specified task. PILOT commands are called by a command label usually one or two letters in length, followed by a colon and an action statement.

The TYPE command (T:) prints a message on the screen. The message is displayed on the line below the command. Here is an example:

```
T: MY COMPUTER DOES NOT HAVE EARS  
MY COMPUTER DOES NOT HAVE EARS
```



The SOUND command (SO:) generates up to four voices of musical tones. Each voice generates numbered tones ranging from 1 to 31. A tone of 0 means silence or a rest. An example of the SOUND command is:

```
SO: 1,5,8,11
```

The GRAPHICS command (GR:) generates lines, figures, or colors in graphics mode. The command to open the graphics screen is GR: followed by an action statement (i.e., DRAW 10, FILL 40). The command to clear the graphics screen is GR: CLEAR. To exit graphics mode and return to standard text mode, type: **GR: QUIT**.

SECTION 2 QUIZ

1. If a cartridge isn't inserted into the computer, the computer can be used only as a _____. (Hint: try it and see. Make sure the computer's cover is closed.)
2. An input line is ended by pressing the _____ key.
 - (a) SYSTEM RESET
 - (b) CLEAR
 - (c) RETURN
 - (d) ESCAPE (ESC)
3. To clear the screen you should _____.
 - (a) Turn off the television screen
 - (b) Turn off the computer
 - (c) Press the **SYSTEM RESET** key
 - (d) Press the **SHIFT** and **BREAK** keys.
 - (e) Press the **CLEAR** key
 - (f) Press the **SHIFT** and **CLEAR** keys.
4. The unshifted **DELETE BACK S** key is used to _____.
5. What is a command? _____.
6. Match the commands with their command labels:

SO:	GRAPHICS
T:	SOUND
GR:	TYPE
7. The SOUND command generates musical _____. Each is assigned a number from _____ to _____.
8. To turn on and clear the graphics screen, type: _____.
To turn off the graphics screen and return to text mode, type: _____.
9. The _____ command displays a message on the screen.
10. What is a cursor? _____.

ANSWERS

1. Memo pad
2. (c)
3. (f)
4. Backspacing the cursor to the mistake in an input line for correction
5. A command is an instruction to the computer to carry out a specified task.
6.

SO:	GRAPHICS
T:	SOUND
GR:	TYPE
7. Tones, tone, 0, 31
8. GR: CLEAR, GR:QUIT
9. TYPE (T:)
10. The cursor is the square on the screen that marks your place.

WRITING A PILOT PROGRAM

CONCEPTS INTRODUCED

Immediate and Deferred Commands
 What Is a Program?
 Executive Commands
 The RUN Command
 The LIST Command
 The NEW Command
 The AUTO Command
 More Commands
 The TYPE Command, T:
 The ACCEPT Command, A:
 The REMARK Command, R:
 The END Command, E:
 Correcting and Changing Your Programs
 Erase a Line
 Insert a Line
 Replace a Line
 Line Renumbering
 The RENUMBER Command, REN
 Fancy Renumbering With REN

IMMEDIATE AND DEFERRED COMMANDS

When the computer receives a command, one of two things happen:

- It is obeyed immediately.
- It is remembered and obeyed at a later time.

Commands obeyed instantaneously when you press the **RETURN** key are called (*immediate commands*). The examples of the T:, SO:, and GR: commands in Section 2 are immediate commands. Commands stored in memory to be obeyed later are called *deferred commands*.

WHAT IS A PROGRAM

A *program* is a sequence of instructions that directs a computer to perform a desired operation. A program is composed of *statements*. A statement is a command, or set of commands, that is preceded by a *line number*.

Line numbers help the computer differentiate between immediate and deferred statements. Line numbers also keep the statements in numerical order. A *statement without a line number is an immediate statement*. A *statement with a line number is a deferred statement*, and is part of a program.

■ A line number must be an integer between 0 and 9999. Duplicate line numbers are not allowed. ■

Here is an example of a program:

```
10 T: HELLO,  
20 T: I AM HAVING FUN  
30 T: LEARNING TO  
40 T: TALK IN  
50 T: PILOT
```

Each TYPE command is preceded by a line number. The TYPE messages are stored in memory in sequential order from 10 to 50 to be displayed at a later time.

Following is a new program using the TYPE command. Press the **RETURN** key at the end of each statement.

```
10 T: MY  
20 T: COMPUTER HAS  
30 T: A NAME.  
40 T: IT IS  
50 T: FRED.
```

■ Always press the **RETURN** key at the end of each line to send the line to the computer's memory. ■

What happened when you pressed **RETURN** at the end of line 50?

Nothing? Well, that's what is supposed to happen! Nothing! Unlike immediate commands, deferred commands are not obeyed when the **RETURN** key is pressed. To execute a program (a sequence of deferred commands) you must type:

```
RUN RETURN
```

Go ahead! Type **RUN**, press **RETURN** and watch your program go!

```
MY  
COMPUTER HAS  
A NAME.  
IT IS  
FRED.  
READY  
■
```

EXECUTIVE COMMANDS

Executive commands are immediate commands that tell the computer *what to do with your program*. EXECUTIVE COMMANDS DO NOT HAVE LINE NUMBERS. Here are four executive commands:

```
RUN  
LIST  
NEW  
AUTO
```

THE RUN COMMAND

The **RUN** command tells the computer to execute, or perform, the program currently in the computer's memory. When you're ready to execute a program, type: **RUN** and press **RETURN**.

Try it. Your screen should still have the example program displayed on it. Type: **RUN** and press **RETURN**. Watch the computer display your programmed messages. Your screen before and after the RUN command should look like this:



Figure 3-1 Program Run

When the RUN command is issued (followed by a **RETURN** of course), the computer automatically *clears the screen* before executing the program. When program execution is completed, the screen displays the READY message and the cursor.

■ Because the computer can only remember one program at a time, every time you issue the RUN command the computer executes the program currently stored in memory. ■

THE LIST COMMAND

The LIST command displays the program statements currently in the computer's memory.

To display our example program clear the screen and on a new line, type:

LIST **RETURN**

The program statements, listed in sequential order, are displayed on the screen.

```
LIST
10 T:MY
20 T:COMPUTER HAS
30 T:A NAME.
40 T:IT IS
50 T:FRED.
```

```
READY
■
```

Once listed, you can check the program for mistakes, and then change, add, or delete lines from the program. The various ways to erase, replace, or insert lines from within a program will be discussed at the end of this section.

The LIST command lists entire programs or portions of a program, depending upon how you enter the LIST command:

LIST 10,50	displays the entire program
LIST 10	displays line 10
LIST 30	displays line 30
LIST 20,40	displays lines 20 through 40
LIST 10,20	displays lines 10 through 20

■ When entering two line numbers, the first line number must be smaller than the second line number. If you list a line number not actually in the program, your ATARI computer responds with a blank line. ■

Let's try some examples of the various forms of the LIST command using the program example.

First, clear the screen.

■ To clear the screen, hold down the **SHIFT** key while pressing the **CLEAR** key. ■

Type: **LIST 20** **RETURN**

```
LIST 20
20 T:COMPUTER HAS
```

```
READY
■
```

Clear the screen. Type: **LIST 10,30** **RETURN**

```
LIST 10,30
10 T:MY
20 T:COMPUTER HAS
30 T:A NAME.
```

```
READY
■
```

Clear the screen. Type: **LIST 40,50** **RETURN**

```
LIST 40,50
40 T:IT IS
50 T:FRED.
```

```
READY
■
```

Are you beginning to understand how this works?

Clear the screen. Type: **LIST 15** **RETURN**

LIST 15

READY

■

Because there is not a line number 15 in the program, a blank line is displayed.

THE NEW COMMAND

*The NEW command erases the current program from the computer's memory, but not from the screen. Be **VERY CAREFUL** when using the NEW command; it is easy to lose your program accidentally, and there is no way to recover it.*

To erase a program from memory, type:

NEW **RETURN**

Let's test the NEW command on the example program.

Type: **NEW** **RETURN**

NEW

READY

■

Try to LIST the example program again.

NEW

READY

LIST

READY

■

Wow! It works! We have erased the program from memory!

■ Always erase an old program with the NEW command before entering a new program. Otherwise, you'll end up with a mixture of both. ■

THE AUTO COMMAND

The AUTO command automatically assigns line numbers to program statements as they are entered. This convenient command replaces the task of sequential line numbering as long as you enter the statements in the proper order.

To enter AUTO mode, type **AUTO** and press **RETURN**. The screen will turn gold and retain any display currently on the screen.



Figure 3-2 AUTO Mode

To leave AUTO mode, enter a blank line and press the `RETURN` key.

In AUTO mode you do **not** type line numbers. Anything entered is treated as a program statement, and therefore executive commands may not be entered while in AUTO mode. Of course, if the computer doesn't understand your input it still responds with the `*** WHAT'S THAT? ***` message.

Let's enter a program in AUTO mode. Type in `NEW`, press the `RETURN` key, and enter:

```
AUTO
T:HAIL!
T:HAIL!
T:THE GANG'S ALL HERE!
```

■ Note that you **do not type line numbers** when in AUTO Mode. ■

Although your input looks like three immediate mode statements, they are actually deferred statements. This is evident because nothing happens when you press the `RETURN` key.

Once you've entered the entire program, you must exit AUTO Mode before you can issue a RUN, LIST, or NEW command. If you try, the computer responds with an `*** IMMEDIATE ONLY ***` message. This means that *executive commands can be executed only in immediate or programmed mode, e.g., when the screen is blue.*

To exit AUTO Mode, press the `RETURN` key or enter a blank line.

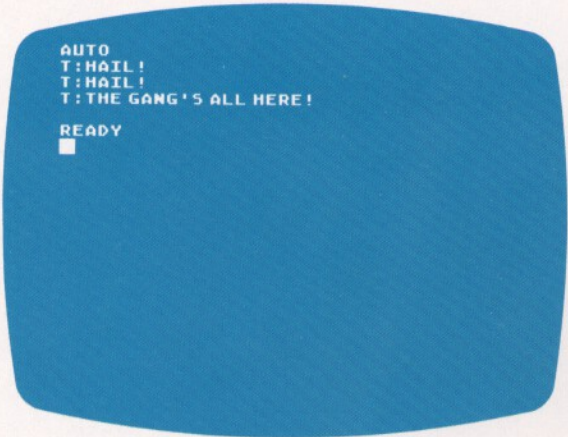
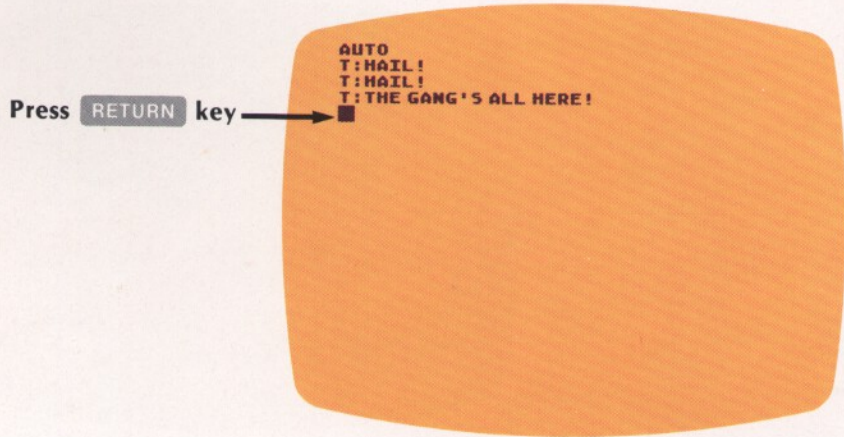
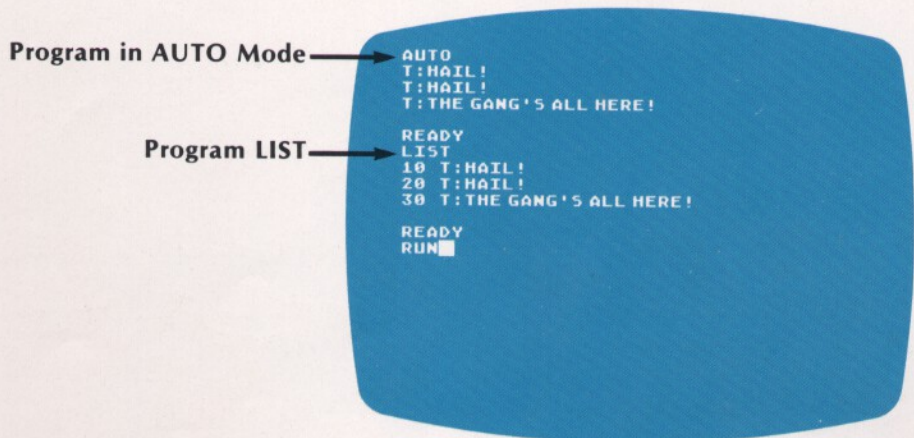


Figure 3-3 Return to Text Mode From AUTO Mode

Now you can enter any executive command or change the program as you wish. LIST and RUN the program:



Program RUN



```
HAIL!  
HAIL!  
THE GANG'S ALL HERE!  
READY  
█
```

Figure 3-4 Program List and Run

MORE COMMANDS

THE TYPE COMMAND, T:

Section 2 explained how to use the TYPE command (T:) in immediate mode. The TYPE command may also be used as a deferred command within a program by assigning a line number to it:

```
10 T: THIS IS AN EXAMPLE OF THE TYPE COMMAND
```

■ **Continuation of the T: Command:** When typing the T: command several times in a row, a solitary colon, :, can be used in place of the command label. The colon **inherits** the command label from the previous line. ■

Thus, type the program:

```
10 T: HAIL!  
20 T: HAIL!  
30 T: THE GANG'S ALL HERE!
```

It can also be written as:

```
10 T: HAIL!  
20 : HAIL!  
30 : THE GANG'S ALL HERE!
```

with the same results.

THE ACCEPT COMMAND, A:

The ACCEPT command accepts input from the keyboard during program execution. When the computer encounters the ACCEPT command it will stop and wait for input from the keyboard. The cursor will reappear and wait for the input. Until the computer receives input, nothing will happen.

The format for a simple ACCEPT command is **A:**.

The ACCEPT command usually follows a TYPE command displaying a message telling you what to input. Look at the examples below:

```
10 T: WHAT IS YOUR NAME?  
20 A:  
30 T: THAT IS A NICE NAME.
```

```
10 T: WHAT COLOR ARE YOUR SHOES?  
20 A:  
30 T: DO THEY MATCH YOUR SHIRT?  
40 A:
```

This is what happens when you run the first program:

Program



```
10 T: WHAT IS YOUR NAME?  
20 A:  
30 T: THAT IS A NICE NAME!  
RUN
```

Program RUN

Cursor waits for your input



```
WHAT IS YOUR NAME?  
█
```

You enter input

Computer's response

```
WHAT IS YOUR NAME?  
ALICE  
THAT IS A NICE NAME!  
READY  
█
```

Figure 3-5 NAME Program Run

A RUN of the second program looks like this:

Program

```
10 T:WHAT COLOR ARE YOUR SHOES?  
20 A:  
30 T:DO THEY MATCH YOUR SHIRT?  
40 A:  
RUN█
```

Program RUN

Cursor waits for your input

```
WHAT COLOR ARE YOUR SHOES?  
█
```

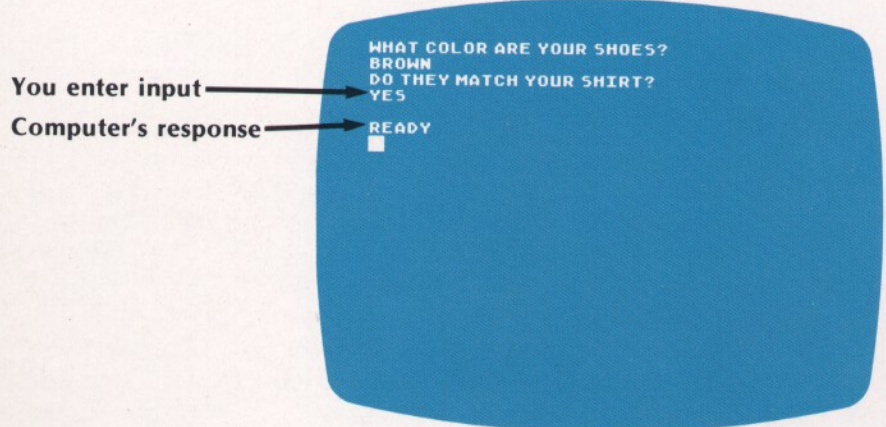
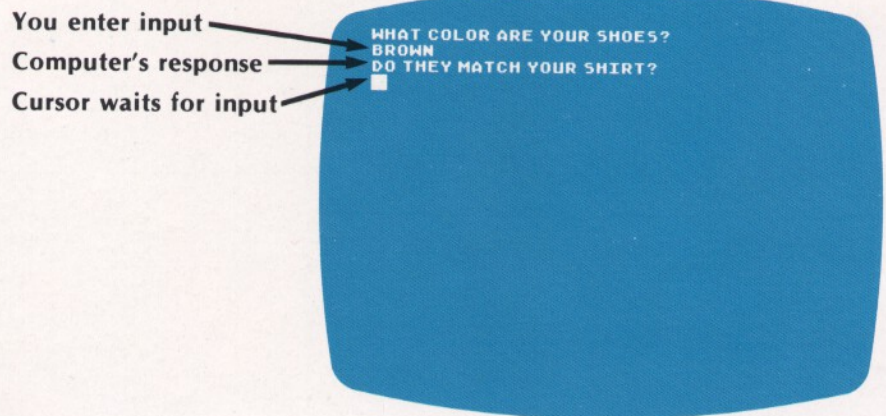


Figure 3-6 CLOTHES Program Run

Do you understand how the TYPE and ACCEPT commands work together?

- Always precede the ACCEPT command with a TYPE command to give the user a hint about what should be input. ■

Let's write a new program. Type:

```

10 T: MY NAME IS FRED.
20 T: WHAT IS YOUR NAME?
30 A:
40 T: I AM A COMPUTER.
50 T: WHAT ARE YOU?
60 A:
70 T: I AM GLAD TO MEET YOU!
80 T: HAVE A NICE DAY!
  
```

Enter **RUN** and press **RETURN** to execute the program.

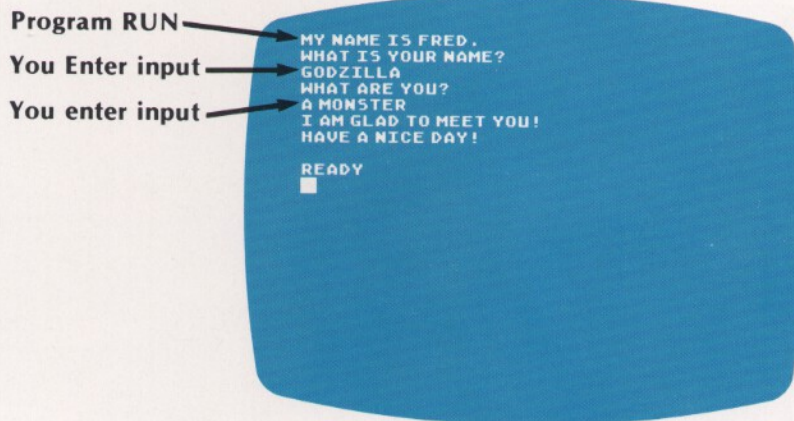


Figure 3-7 TYPE and ACCEPT Commands: Program Run

Now that you know how to use two interactive commands, you can write question-and-answer programs and run them for your friends!

■ **Continuing Text With the Backslash Character:** PILOT uses a special character that keeps the cursor from dropping to the next line. This character is the backslash, \, produced by pressing the **SHIFT** key and the \ + key. When the computer encounters a backslash at the end of a TYPE command, the cursor stops and waits ON THE PRESENT LINE instead of doing a carriage return. Any following text will be displayed from that point. The program below uses the backslash:

```
10 T: THIS IS A VERY AUSPICIOUS OCCASION, \
20 T: MY FRIENDS. IT HAS COME TO MY ATTENTION \
30 T: THAT THERE WILL BE NO MORE FREE \
40 T: CAKE AND ICE CREAM!! \
```

A RUN of this program produces this result:

```
THIS IS A VERY AUSPICIOUS OCCASION, MY
FRIENDS. IT HAS COME TO MY ATTENTION THAT
THERE WILL BE NO MORE FREE CAKE AND ICE
CREAM!!
```

```
READY
■
```

The backslash is especially useful when using a TYPE statement to prompt input from a subsequent ACCEPT command. By placing a backslash at the end of the TYPE message and prior to an ACCEPT command, the TYPE message and the ACCEPT input will appear on the same display line.

Type **NEW** and press **RETURN** before entering the following program:

```
10 T: MY NAME IF FRED.
20 T: WHAT IS YOUR NAME? \
30 A:
40 T: I AM A COMPUTER.
50 T: WHAT ARE YOU? \
60 A:
70 T: I AM GLAD TO MEET YOU! \
80 T: HAVE A NICE DAY!
```

Below is the same program you entered earlier, except that backslash characters have been added to the end of lines 20, 50, and 70. RUN the program and look at what happens:

```
MY NAME IS FRED .
WHAT IS YOUR NAME? CLEOPATRA
I AM A COMPUTER .
WHAT ARE YOU? A DROID
I AM GLAD TO MEET YOU! HAVE A NICE DAY!
```

```
READY
■
```

See how easy it is to put both TYPE and ACCEPT input and output on the same line by adding just one character? For practice, make up some question-and-answer programs for yourself using the backslash.

■ Always type a space before the backslash character, to prevent the words preceding and following the backslash from running together like this: WHAT IS YOUR NAME? CLEOPATRA. ■

THE REMARK COMMAND, R:

The REMARK command lets you insert remarks into your program to make it easier to understand. Remarks are usually titles or little notes explaining what the program does and how it works. The computer ignores REMARK commands, so they can be placed anywhere in a program.

The format for the REMARK command is:

R: *anything*

Look at the REMARK commands in the following program:

```
10 R: *****
20 R: *                               *
30 R: *           THE NOTHING PROGRAM           *
40 R: *                               *
50 R: *****
60 T: This program does NOTHING
70 R: That's for sure! This is the END of the program!
```

■ **Upper- and lowercase letters:** PILOT displays both upper- and lowercase letters. To type lowercase letters, press the **CAPS LOWR** key once. Any alphabetic characters typed after the **CAPS LOWR** key has been pressed will be displayed in lowercase. To type uppercase letters, hold down the **SHIFT** key while pressing the **CAPS LOWR** key. Alphabetic characters typed after the **SHIFT** and **CAPS LOWR** keys are pressed WILL ALL BE UPPERCASE. ■

Upon program execution, the screen looks like:

```
This program does NOTHING
```

```
READY
■
```

The remarks don't show up because they are ignored by the computer during program execution. REMARK commands are only displayed when the program is LISTed.

Remarks can also be placed on the same statement lines as commands by using the **␣** character *instead of* the REM command. The **␣** character must follow the command and precede the remark:

command **␣** *remark*

Examples:

10 A: #N **␣** ACCEPT A NUMBER

50 T: \$NAME **␣** DISPLAY A NAME

THE END COMMAND, E:

The end of a program isn't necessarily the last line of a program. PILOT programs can be terminated at any point with the END command.

The format for the END command is: **E:**

Note: PILOT assumes that the last line is the end of a program unless an END command is present elsewhere.

CORRECTING AND CHANGING YOUR PROGRAMS

ERASE A LINE

To erase a line from a program, *type in the line number and press the* **RETURN** *key.* This deletes the line from the program, in memory, although the line remains on the screen until the program is LISTed again.

Enter this example program:

```
10 T: ROSES ARE RED,  
20 T: VIOLETS ARE BLUE,  
30 T: SUGAR IS SWEET,  
40 T: AND SO ARE YOU!
```

RUN the program. It should look like this:

```
ROSES ARE RED,  
VIOLETS ARE BLUE,  
SUGAR IS SWEET,  
AND SO ARE YOU!
```

READY
■

Let's erase line 30. Type: **30** and press **RETURN**. LIST the program again.

```
30  
LIST  
10 T: ROSES ARE RED,  
20 T: VIOLETS ARE BLUE,  
40 T: AND SO ARE YOU!
```

READY
■

Line 30 is erased from the program. Type: **LIST 30** and press **RETURN**.

```
READY
LIST 30
```

```
READY
■
```

Nothing but a blank line appears! Line 30 is gone forever. If you execute the program, this is what you should get:

```
ROSES ARE RED,
VIOLETS ARE BLUE,
AND SO ARE YOU!
```

```
READY
■
```

INSERT A LINE

New lines can be inserted anywhere in a program by assigning a new line number to a statement. The new number must be given a number between the line number preceding and the line number following the place where the new line will be inserted. LINE NUMBERS CANNOT BE INSERTED IN AUTO MODE.

■ PILOT always rearranges program statements so that their line numbers are in numerical order. ■

Clear the screen and LIST the program:

```
LIST
10 T:ROSES ARE RED,
20 T:VIOLETS ARE BLUE,
40 T:AND SO ARE YOU!
```

```
READY
■
```

Now, let's reinsert the line removed during the previous example. To insert the statement T: SUGAR IS SWEET, between 20 T: VIOLETS ARE BLUE, and 40 T: AND SO ARE YOU!, we must pick a line number between 20 and 40. In other words, by assigning T: SUGAR IS SWEET, any line number from 21 to 39, the statement will be inserted between lines 20 and 40.

Type: 23 T: SUGAR IS SWEET, press **RETURN**, and LIST the program again:

```
READY
23 T: SUGAR IS SWEET,
LIST
10 T:ROSES ARE RED,
20 T:VIOLETS ARE BLUE,
23 T: SUGAR IS SWEET,
40 T:AND SO ARE YOU!
```

```
READY
■
```

RUN the program:

```
ROSES ARE RED,
VIOLETS ARE BLUE,
SUGAR IS SWEET,
AND SO ARE YOU!
```

```
READY
■
```

Voila! The program is back to its original order.

REPLACE A LINE

Replacing a line is a short-cut method of deleting a line and inserting another in its place.

Clear the screen and LIST the example program again. Let's change the rhyme by replacing line 23; T: SUGAR IS SWEET, to; T: DAFFODILS ARE PRETTY,. This can be done in two ways.

First, you can delete line 23 by typing **23** and pressing **RETURN**. Next, you type in the new line: **23 T:DAFFODILS ARE PRETTY**, and press **RETURN**. Then LIST the program:

```
23
23 T:DAFFODILS ARE PRETTY,
LIST
10 T:ROSES ARE RED,
20 T:VIOLETS ARE BLUE,
23 T:DAFFODILS ARE PRETTY,
40 T:AND SO ARE YOU!
```

READY



You have replaced the old line 23 with a new line 23.

The second method of line replacement is to type the new line 23 on a new line, without first deleting the old line 23. When the computer receives the new line 23, it will automatically push the old line 23 out of memory and replace it with the new line 23. The old line 23 is pushed out of memory because the computer doesn't allow duplicate line numbers.

Let's change line 23; T: DAFFODILS ARE PRETTY, back to the original line 23; T: SUGAR IS SWEET,.

Clear the screen and LIST the program. Type: **23 T: SUGAR IS SWEET**, press **RETURN** and LIST the program again:

```
LIST
10 T:ROSES ARE RED,
20 T:VIOLETS ARE BLUE,
23 T:DAFFODILS ARE PRETTY,
40 T:AND SO ARE YOU!
```

READY

```
23 T: SUGAR IS SWEET,
LIST
10 T:ROSES ARE RED,
20 T:VIOLETS ARE BLUE,
23 T: SUGAR IS SWEET,
40 T:AND SO ARE YOU!
```

READY



■ Remember that the line change is not displayed on the screen until the program is LISTed again. ■

LINE RENUMBERING

PILOT has a renumber command that automatically renumbers the line numbers of an entire program. This is useful when you need to insert a statement between two statements with consecutive line numbers.

Type: **NEW** and press **RETURN** to clear the computer's memory. Enter the following program:

```
5 T: WHAT
6 T: A
7 T: PROGRAM
```

A program RUN displays:

```
WHAT
A
PROGRAM

READY
■
```

Now consider changing the above program to display this instead:

```
WHAT
A
SIMPLE
PROGRAM

READY
■
```

How would you make the change? You could change the last half of the program by line deletion and replacement to make enough room to enter a **T: SIMPLE**, or you could renumber the program with the renumber command.

The **RENUMBER** Command, **REN**

The format for a simple **REN** command is: **REN**

REN by itself automatically renumbers the entire program starting with line number 10, and then counting by tens for as many lines as are in the program. This creates enough space to insert extra lines.

Type: **REN** and press **RETURN**. Now, **LIST** the program. You should see:

```
REN

READY
LIST
10 T: WHAT
20 T: A
30 T: PROGRAM

READY
■
```

You now have room to insert an extra line.

Type: **25 T: SIMPLE** and press **RETURN**. LIST the program again. You should see:

```
READY
25 T: SIMPLE
LIST
10 T: WHAT
20 T: A
25 T: SIMPLE
30 T: PROGRAM
```

```
READY
■
```

Fancy Renumbering With REN

The following format for the REN command allows you to choose the starting line number of the program. It also allows you to renumber a program with an increment other than 10.

REN *starting line number*

REN *starting line number, increment*

To renumber the program so that it begins with line number 100, type:

```
REN 100 RETURN
```

A program LIST displays:

```
REN 100

READY
LIST
100 T: WHAT
110 T: A
120 T: SIMPLE
130 T: PROGRAM
```

```
READY
■
```

To renumber a program with an increment other than 10, follow the starting line number with a comma and the increment value. To renumber the program by 2, type:

```
REN 100,2 RETURN
```

A program LIST displays:

```
REN 100,2

READY
LIST
100 T: WHAT
102 T: A
104 T: SIMPLE
106 T: PROGRAM
```

```
READY
■
```

Beware of choosing too high a starting line number or too large an increment. REN will produce an error if a line number tries to exceed 9999.

SECTION 3 SUMMARY

If you enter: **REN 9000,500**, the computer responds with:

```
REN 9000,500  
*** LINE # ? ***
```

A LISTing now shows:

```
REN 9000,500  
*** LINE # ? ***
```

```
LIST  
9000 T:WHAT  
9500 T:A  
104 T:SIMPLE  
106 T:PROGRAM
```

```
READY  
■
```

Lines 104 and 106 were left over because their "new" line numbers would exceed 9999. To correct this, just REN again with a lower starting number and increment.

An *immediate command* is a command that is obeyed when the **RETURN** key is pressed. Commands that are stored in memory, to be obeyed upon a RUN command, are called *deferred commands*.

A *program* is a sequence of deferred instructions that direct the computer to perform a desired operation. A program is made up of one or more *program statements*. A program statement is a command preceded by a line number. *Line numbers* keep the program statements in a specific order.

Executive commands, such as RUN, LIST, NEW, and AUTO, tell the computer what to do with a program. They are immediate commands.

RUN is an executive command. It tells the computer to execute the PILOT program currently in the computer's memory.

LIST displays the program currently in memory. It does not execute the program. The whole program, or specified lines may be listed with this format:

LIST	lists entire program
LIST line x	lists line x of the program
LIST line x, line y	lists lines x through y of the program.

The NEW command erases the current program from the computer's memory. Once a program is deleted with NEW, it can't be retrieved.

The AUTO command puts the system into *Automatic Line Numbering Mode*, distinguished by a gold screen. In AUTO mode, program statements are input without line numbers. The line numbers are automatically assigned as the statements are input. The line numbers are not displayed until the program is LISTed in immediate or program mode (blue screen). Executive commands may not be issued in AUTO mode. To return from AUTO mode to immediate mode, press the **RETURN** key alone, entering a blank line.

The TYPE command (T:) displays text messages. In successive TYPE commands, the T: can be replaced by a single colon. The colon *inherits* the previous TYPE command label.

The ACCEPT command is used in conjunction with the TYPE command. The ACCEPT command, A:, allows input from the keyboard to be entered during program execution. The ACCEPT command delays program execution until input is received from the keyboard, ending with the **RETURN** key.

To delete a line from a program, type the line number and press the **RETURN** key. This erases the line from the program in memory, but not from the screen display until the program is LISTed.

To insert a line into a program, enter it using a new line number. The new number must fall between the line numbers where the line is to be inserted. The screen will not show the inserted line in its correct position until the program is LISTed.

There are two methods to replace a line. Delete the old line and insert a new line using the same line number. Or, simply assign the new line the old line number. The change will not be shown on the screen until the program is LISTed.

SECTION 3 QUIZ

1. A program is made up of _____ commands that are executed when issued a RUN command.
2. A program statement is _____.
3. RUN, LIST, NEW, and AUTO are all examples of _____ commands. These commands are executed only in _____ mode.
4. The RUN command tells the computer _____.
5. The LIST command displays the program in memory and then executes the program. TRUE or FALSE?
6. To LIST lines 20, 30, 40, 50, and 60 of a program, you would enter: _____.
7. The NEW command erases a program from the screen but not from memory. TRUE or FALSE?
8. The AUTO command lets the computer _____ to program statements.
9. In AUTO mode the color of the screen is: _____
 - (a) Blue
 - (b) Black
 - (c) Gold
 - (d) Red
 - (e) Green
10. To exit AUTO mode, you _____.
 - (a) Press the **BREAK** key.
 - (b) Press the **RETURN** key.
 - (c) Type: **AUTO: QUIT**
 - (d) Type: **AUTO: CLEAR**
 - (e) Press the **SELECT** key.

-
11. The TYPE command is often used in conjunction with the ACCEPT command. Why? _____.
12. Match the following:
- | | |
|----------------|---|
| Insert a line | Enter line number and press the RETURN key |
| Replace a line | Assign a statement a new line number |
| Delete a line | Use the same line number on a new or different line |
13. The computer ignores REMARK commands. REM commands are useful because they allow you to _____.
14. The _____ command lets you terminate a program before the last line of a program.
15. Typing a line number and pressing the **RETURN** key _____ a line.
16. To insert a new line you must _____.
17. The REN command will renumber a program by any increment. TRUE or FALSE?

ANSWERS

1. Deferred
2. A command preceded by a line number.
3. Executive commands; immediate mode.
4. Execute the program currently in the computer's memory.
5. FALSE. LIST displays the current program on the screen.
6. LIST 20,60 `RETURN`
7. FALSE. NEW erases the program from *memory*, not the screen.
8. Assign line numbers
9. (c) Gold
10. (b) Press the `RETURN` key.
11. The TYPE command precedes the ACCEPT command to tell the user what type of data to enter.
12.

Insert a line	Enter line number and press the <code>RETURN</code> key
Replace a line	Assign a statement a new line number
Delete a line	Use the same line number on a new or different line.
13. Document your programs or add titles and notes to explain the program.
14. END (E:)
15. Erases
16. Pick a line number in the correct sequential position, type the line number, the program statement, and press the `RETURN` key.
17. TRUE

DECISION-MAKING PROGRAMS

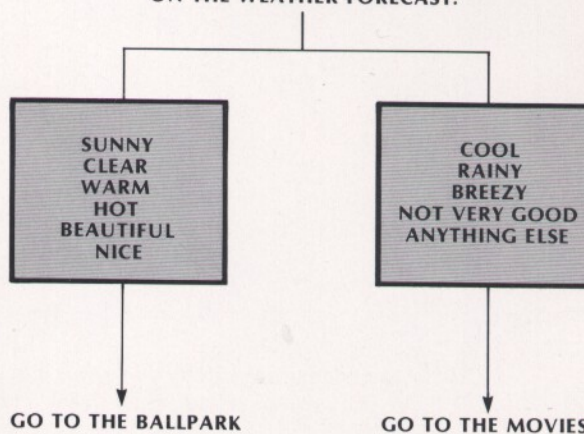
CONCEPTS INTRODUCED

- Matching for Clues
 - The Function of the ACCEPT Command, A:
 - The MATCH Command, M:
- Conditional Commands
 - The TYPE IF YES and TYPE IF NO Commands, TY:, TN:
 - The END IF YES and END IF NO Commands, EY:, EN:
- Section 4 Summary
- Section 4 Quiz
- Advanced Decision-Making Techniques
 - Separating MATCH Strings With Vertical Bars

PILOT programs can use keyboard input to make decisions. Decisions are made in PILOT in much the same way the person in the following sketch made a decision to go to the ballpark.

TO GO OR NOT TO GO

IDEA: YOU WANT TO GO TO THE BALLPARK TOMORROW. YOU MUST MAKE YOUR PLANS TODAY. YOUR DECISION DEPENDS ON THE WEATHER FORECAST.



The advantage of using the PILOT language over other programming languages lies in the ease with which it allows you to perform complex pattern-matching on user input and then to make decisions relative to the presence or absence of a "match." With just three commands (TYPE, ACCEPT, and MATCH) you can create a reasonably sophisticated, interactive dialogue program. This section shows you how to write educational, instructional, and just plain fun PILOT programs using pattern-matching decisions.

MATCHING FOR CLUES

The MATCH command functions like a detective. MATCH searches the keyboard input received from an ACCEPT command for a specific pattern or patterns. If a matching pattern is found, PILOT remembers that there is a match. If no pattern is found, PILOT remembers that there isn't a match.

THE FUNCTION OF THE ACCEPT COMMAND, A:

The ACCEPT command has a special function: to retrieve and retain your input so it can be searched for a specific pattern by the MATCH command. Input from the keyboard is retrieved by the ACCEPT command in the format:

A: *your input*

Your input is called a **string**.

■ A string is a series of letters, numbers, or symbols. Words, sentences, and phrases are all strings.

HOW ARE YOU?	is a string
KEITH	is a string
APRIL 1, 1981	is a string ■

Anytime an ACCEPT command is executed, your input string is placed in a specified area called the **accept buffer**. Think of the accept buffer as a compartment in memory that "holds" your input string.



Figure 4-1 Concept of the Accept Buffer

Any lowercase characters in your input string are converted to uppercase letters before they are searched by a MATCH command.

THE MATCH COMMAND, M:

The MATCH command lists the pattern or patterns to be searched for and matched in the accept buffer. Each pattern is also called a *string*.

The format for the MATCH command is: **M:** *string*

If a MATCH command specifies several strings *each string must be separated by a comma*.

M: *string, string...*

Here are some examples:

M: TABLE

M: YELLOW,BLUE,RED

M: 1,2,3,4

M: ALL CATS ARE FURRY, ALL DOGS HAVE TAILS

M: ,

The MATCH command followed by only a comma will match ANY input.

■ Spaces are important characters in MATCH strings. For example:

M: A

matches any word in the input string beginning with the letter A. Because a space always precedes the beginning of a word in the accept buffer, the combination of a preceding space and the letter A tells the computer to *match any word beginning with the letter A*.

Examples:

ATARI is a match

HOT AIR BALLOON is a match

BAGEL is NOT a match

However, this MATCH command:

M:A

matches input that has the letter A *anywhere* in the string:

EASY is a match

JANUARY is a match

AIRPLANE is a match ■

■ Often you may want a MATCH command to search for a fragment of a word or string, or for the ending letter(s) of a word. For instance, to search an input string for a word that rhymes with *moose*, your MATCH command might look like this:

M: OOSE __,UCE __

The underscore character, __, also represents a space at the end of a string. ■

■ To type an underscore character, __, hold down the **SHIFT** key and press the **_ _** key. ■

■ An underscore character placed at the *end* of a MATCH string means that a trailing space is part of the string and must be part of the string to produce a match. In this way, you can match for the ending letter, word, or fragment of a word in an input string.

Using the M: OOSE __, UCE __ example, any input string that ends with OOSE or UCE produces a match:

GOOSE	is a match
LOOSE	is a match
CABOOSE	is a match
DUCE	is a match
TRUCE	is a match ■

CONDITIONAL COMMANDS

The MATCH command searches the input string received from an ACCEPT command for a matching string. The computer remembers if there is a match or no match. PILOT uses the MATCH command followed by conditional commands to perform specific instructions conditional upon a match or no match. A MATCH command must be preceded by a conditional command.

Here are some of PILOT's conditional commands:

TY: or Y:	TYPE IF YES command
TN: or N:	TYPE IF NO command
EY:	END IF YES command
EN:	END IF NO command

The TYPE IF YES and TYPE IF NO Commands TY:, TN:

The TYPE command can be used with *yes* or *no* conditioners in the following formats:

TY: *message*
Y: *message*
TN: *message*
N: *message*

If a match is found and there is a command with a Y conditioner, it will be executed (e.g., TY: *message*). If a match is NOT found and there is a command with a N conditioner it will be executed (e.g., TN: *message*). Both the Y and N conditioners may be used for a single MATCH command as long as they are not in the same statement. But, if both the Y and N conditioners follow a single MATCH command, *only one of the commands will be executed*; the other one is bypassed.

Here is a sample program to demonstrate the interaction of the M:, TY: and TN: commands:

```
1 R: HISTORY PROGRAM
10 T:
20 T:
30 T: Who was the first president of \
40 T: the United States? \
50 A:
60 M: GEORGE WASHINGTON
70 T:
80 TY: That's right!!
90 TN: No. The first president of the \
100 TN: United States was GEORGE \
110 TN: WASHINGTON.
```

If the correct answer (GEORGE WASHINGTON) is entered when the program is executed, the program executes the TY: command to display:

```
Who was the first president of the
United States? GEORGE WASHINGTON
```

```
That's right!!
```

```
READY
■
```

If an incorrect answer is entered, the program executes the TN: command to display:

```
Who was the first president of the
United States? UNCLE SAM
```

```
No. the first president of the United
States was GEORGE WASHINGTON.
```

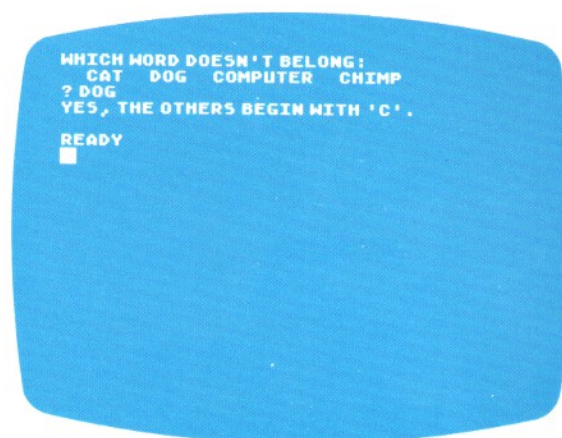
```
READY
■
```

Try another one. Type NEW and enter the following program:

```
10 T: WHICH WORD DOESN'T BELONG:
20 T: CAT DOG COMPUTER CHIMP
30 T:? \
40 A:
50 M: DOG
60 TY: YES, THE OTHERS BEGIN WITH 'C'.
70 M: COMP
80 TY: YES, THE OTHERS ARE ANIMALS.
90 M: COMP,DOG
100 TN: ARE YOU SURE ABOUT THAT?
```

Run the program with each of the four words in line 20 as input.

Here are three RUNS of the same program:



```
WHICH WORD DOESN'T BELONG:
  CAT DOG COMPUTER CHIMP
? COMPUTER
YES, THE OTHERS ARE ANIMALS.
READY
■
```

```
WHICH WORD DOESN'T BELONG:
  CAT DOG COMPUTER CHIMP
? CHIMP
ARE YOU SURE ABOUT THAT?
READY
■
```

Figure 4-2 MATCH Command: Sample Runs

Notice that there are three different possible outputs, depending upon the input string.

What happens if you input DOG AND COMPUTER? Try it and see.

Did your screen display:

```
WHICH WORD DOESN'T BELONG:
  CAT DOG COMPUTER CHIMP
?DOG AND COMPUTER
YES, THE OTHERS BEGIN WITH C'.
YES, THE OTHERS ARE ANIMALS.
READY
■
```

Because DOG AND COMPUTER produces a match in both MATCH statements, both TY: commands are executed.

■ Look closely at line 70:

```
70 M: COMP
```

The MATCH command searched for only the first four letters of COMPUTER. This is advantageous because it allows for spelling mistakes (computer, compater). ■

Putting It All Together

You now possess enough information to understand how to make a program out of the TO GO OR NOT TO GO sketch to decide whether or not to go to the ballpark. Type in the following program:

```
10 T: WHAT IS THE WEATHER LIKE TODAY?  
20 A:  
30 M: SUNNY,CLEAR,WARM,HOT,BEAUTIFUL,NICE,GREAT  
40 TY: YOU SHOULD GO TO THE BALLPARK  
50 TN: YOU SHOULD GO TO THE MOVIES
```

RUN the program with the following ACCEPT inputs:

```
SUNNY AND BRIGHT  
CLEAR AND WARM  
GREAT DAY  
COOL AND BREEZY  
RAINY, CLEARING TOMORROW  
WINDY  
NOT VERY GOOD
```

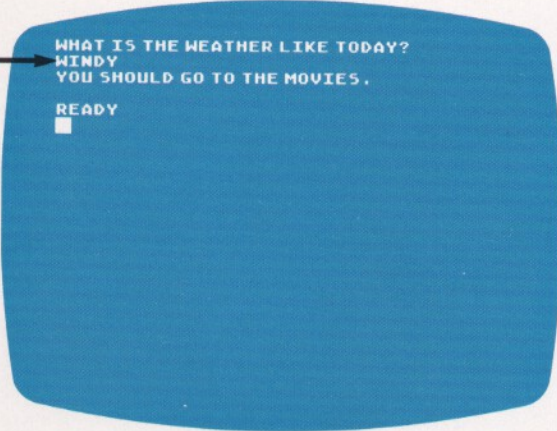
Input: SUNNY →

```
WHAT IS THE WEATHER LIKE TODAY?  
SUNNY  
YOU SHOULD GO TO THE BALLPARK  
  
READY  
■
```

Input: RAINY, CLEARING TOMORROW →

```
WHAT IS THE WEATHER LIKE TODAY?  
RAINY, CLEARING TOMORROW  
YOU SHOULD GO TO THE BALLPARK  
  
READY  
■
```


Input: WINDY



```
WHAT IS THE WEATHER LIKE TODAY?  
WINDY  
YOU SHOULD GO TO THE MOVIES .  
READY  
█
```

Figure 4-3 Sample Runs of the TO GO OR NOT TO GO Program

The MATCH command matched the "CLEAR" in RAINY, CLEARING TOMORROW. However, it doesn't know that the CLEAR is referring to tomorrow's weather, while today's weather is RAINY. You must be very careful about what strings you specify in the MATCH command and the messages you use in the TY: and TN: commands, or else you may get the wrong type of match.

Here is another fun program: a RIDDLE program. Type it in and give it a RUN:

```
10 T: RIDDLE  
20 T: WHAT IS THE LONGEST WORD IN ENGLISH?  
30 A:  
40 M: SMILES  
50 TY: YOU ARE ABSOLUTELY CORRECT!!  
60 TN: SORRY, THE WORD IS 'SMILES!'  
70 T: THERE IS A MILE BETWEEN THE S's.
```

The END IF YES and END IF NO Commands, EY:, EN:

These commands allow the program to be conditionally terminated before the end of a program.

The formats for the END IF YES and END IF NO commands are:

EY:

EN:

Notice that *nothing follows the colon* in the E:, EY:, and EN: commands.

If a match is found and there is an EY: conditional command, the program will terminate. If a match is NOT found and there is an EN: conditional command, the program will terminate.

■ Remember that both an EY: and EN: command may follow a single MATCH command if they are in different program statements. ■

The following program demonstrates the use of the END IF NO command. If the input does not match the match string, the program displays a TN: message and terminates the program with the EN: command. If the input matches, the program continues. The only way to complete the entire program is to answer the three questions correctly:

10 T: THIS PROGRAM MIXES TWO PRIMARY COLORS
20 T: (RED, YELLOW, BLUE)
30 T:
40 T: YOU MUST GUESS THE COLOR!
50 T:
60 T: RED AND YELLOW MAKE: \
70 A:
80 M: ORANGE
90 TY: *** THAT'S RIGHT! TRY ANOTHER ONE! ***
100 TN: *** OOPS! THAT'S NOT RIGHT! ***
110 EN:
120 T:
130 T: YELLOW AND BLUE MAKE: \
140 A:
150 M: GREEN
160 TY: *** THAT'S RIGHT! TRY ANOTHER ONE! ***
170 TN: *** OOPS! THAT'S NOT RIGHT! ***
180 EN:
190 T:
200 T: BLUE AND RED MAKE: \
210 A:
220 M: PURPLE
230 TY: *** THAT'S RIGHT! ***
240 TY:
250 TY: CONGRATULATIONS! YOU GUESSED ALL
260 TY: THREE QUESTIONS CORRECTLY!
270 TN: *** OOPS! THAT'S NOT RIGHT! ***
280 E:

Below is a sample RUN *if* all three questions are answered *correctly*:

```
THIS PROGRAM MIXES TWO PRIMARY COLORS  
( RED, YELLOW, OR BLUE )  
  
YOU MUST GUESS THE COLOR!  
  
RED AND YELLOW MAKE : ORANGE  
*** THAT 'S RIGHT! TRY ANOTHER ONE! ***  
  
YELLOW AND BLUE MAKE : GREEN  
*** THAT 'S RIGHT! TRY ANOTHER ONE! ***  
  
BLUE AND RED MAKE : PURPLE  
*** THAT 'S RIGHT! ***  
  
CONGRATULATIONS! YOU GUESSED  
ALL THREE QUESTIONS CORRECTLY!  
  
READY  
■
```

Here is a sample RUN *if* the first question is answered *incorrectly*:

```
THIS PROGRAM MIXES TWO PRIMARY COLORS  
( RED, YELLOW, OR BLUE )  
  
YOU MUST GUESS THE COLOR!  
  
RED AND YELLOW MAKE : BROWN  
*** OOPS! THAT 'S NOT RIGHT! ***  
  
READY  
■
```

SECTION 4 SUMMARY

As soon as the input did not match the MATCH string, the EN: command terminated the program.

The beauty of the PILOT language lies in its ability to perform complex pattern-matching on user input and then to make decisions relative to the presence or absence of a "match."

The MATCH command searches the keyboard input from an ACCEPT command for a specific pattern. If a matching pattern is found, PILOT remembers that there is a match. If no pattern is found, PILOT remembers that there is no match.

The format for the MATCH command is:

M: *string*

M: *string, string...*

If more than one string is specified in the MATCH command, the strings must be separated by commas.

M:, will match upon any input.

The MATCH command is followed by a conditional command or commands. The TYPE and END commands, combined with a *yes* or *no* conditioner, help PILOT decide what action to take depending upon a match or no match. If a match is found and there is a command with a Y conditioner, it will be executed. If a match is not found and there is a command with a N conditioner, it will be executed.

The TYPE IF YES command (TY:) or (Y:) and TYPE IF NO command (TN:) or (N:) display a message depending upon the success of the match.

TY: *message*

Y: *message*

TN: *message*

N: *message*

The END IF YES command (EY:) and the END IF NO command (EN:) terminate program execution depending upon the success of the match:

EY:

EN:

SECTION 4
QUIZ

1. The MATCH command searches _____ from the _____ command for a matching pattern. The pattern(s) to be matched are specified in the _____ command.
2. Your keyboard input is called a _____.
3. Identify the string(s) below: _____
 - (a) HOWARD MEEDLY
 - (b) AUGUST 31, 1999
 - (c) 12345
 - (d) 12:45
 - (e) a and b
 - (f) b and d
 - (g) All the above
4. A series of strings in the MATCH command must be separated by _____.
5. Depending upon the success of a match, the conditional commands perform a desired operation. If the match is successful, and the Y conditional command is performed, what happens to the N conditional command (if it is present)?

6. Depending upon a yes or no match, the TY: and TN: commands _____.
7. A program can be terminated before the last line of a program depending upon the success of a match and the presence of an END IF NO command. TRUE or FALSE?
8. The underscore character at the end of a MATCH string represents a _____ in the string.

ANSWERS

1. Input (or strings), ACCEPT, MATCH
2. String
3. (g)
4. Commas
5. It is bypassed.
6. Display its message
7. TRUE
8. Space

ADVANCED DECISION- MAKING TECHNIQUES

SEPARATING MATCH STRINGS WITH VERTICAL BARS

If a MATCH command specifies several strings to be matched, *the strings must be separated by commas or vertical bars (|).* If vertical bars are used to separate the MATCH strings, then a vertical bar must follow the colon, preceding each string:

M: | IS | ARE

M: | YELLOW | BLUE | RED

■ To type a vertical bar, hold down the **SHIFT** key and press the | \ key. ■

Vertical bars are especially useful when matching for punctuation marks such as commas, periods, semicolons, colons, and the like:

M: | , | . | ; | : | /

M: | * | +

M: | ,

■ Commas and vertical bars may *not* be used as separators in the same MATCH command. ■



BRANCHING

CONCEPTS INTRODUCED

The Statement Label
 The JUMP Command, J:
 The JUMP IF YES and JUMP IF NO Commands, JY:, JN:
 The TRACE Command
 Section 5 Summary
 Section 5 Quiz
 Advanced Branching Techniques
 The JUMP Command in Immediate Mode, J:
 The JUMP ON MATCH Command, JM:
 Mini-Quiz

So far, all of the programs that you have entered were obeyed in line number order and proceeded from lower line numbers to higher line numbers. The term for this is *sequential program execution*. PILOT allows you to *break the normal sequence of execution* by using something called a *label*.

THE STATEMENT LABEL

A statement label is a way of *naming* a PILOT statement line. The line number is not actually a name. Line numbers do help locate PILOT lines in relation to each other. A label does more: it tells the computer "I'm special!" The computer remembers the statement label's location and its location within the program by the line numbers.

The format for the statement label is:

line number *label name

where: *label name* is any combination of letters and numbers beginning with an asterisk and a letter. Embedded spaces are not allowed.

Examples:

10 *LABEL
 50 *START

THE JUMP COMMAND, J:

The JUMP command tells the computer to break sequential program execution and jump to a labeled statement line.

The format for the JUMP command is:

J: statement label

Examples:

100 J: *LABEL
 250 J: *START
 300 J: *DOAGAIN

- If two or more statements have the same statement label, the JUMP command jumps to the statement label with the lowest line number:

```

10 *FILL
.
.
50 J:*FILL
.
.
100 *FILL

```

Make sure that each JUMP command has a label to jump to. *It is an error to jump to a nonexistent label.* If you try, the program will stop and display a *** WHERE? *** message.

Following is an example program that demonstrates the JUMP command. The program asks you to input colors:

```

10 R: *****
20 R: *   THE COLOR WHEEL   *
30 R: *****
40 T: THIS PROGRAM TESTS HOW
50 T: MANY COLORS YOU KNOW
60 T:
70 T: ENTER A COLOR: \
80 A:
90 *DOAGAIN
100,      T:
110,      T: ENTER ANOTHER COLOR: \
120,      A:
130,      J: *DOAGAIN
140 E:

```

- *Command indentation:* A comma immediately following the line number allows you to indent a PILOT command any number of spaces. Labels, however, may not follow a comma.

This program is intentionally missing some commands that would allow the program to stop by itself. These “missing” commands are omitted because you have not been introduced to them yet. (They will be discussed shortly.) Therefore, this program goes into what programmers often call an *indefinite loop*. In other words, it will not stop until you press the **BREAK** key to stop it. ■

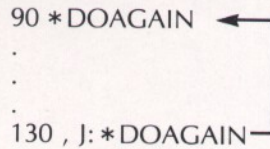
An example RUN of the program looks like this:

```

THIS PROGRAM TESTS
HOW MANY COLORS YOU KNOW
ENTER A COLOR: RED
ENTER ANOTHER COLOR: VIOLET
ENTER ANOTHER COLOR: GREEN
ENTER ANOTHER COLOR: PINK
ENTER ANOTHER COLOR:
120,      A:
*** READY ***

```

The output at the bottom shows what statement was executing if you press the **BREAK** key in mid-execution of a program. As you can see, the program repeats certain statements: it jumps from line 130, J: *DOAGAIN to label *DOAGAIN at line 90, and executes all the statements from 90 to 130 before it jumps to 90 *DOAGAIN again. This repetition of a group of statements is called *looping*. The repeated group of statements is called a *loop*.



THE JUMP IF YES AND JUMP IF NO COMMANDS, JY:, JN:

The JUMP command, in conjunction with the MATCH command, can use *yes* and *no* conditioners to make program decisions. The rules for using conditioners with the JUMP command are the same rules used for the TY:, TN:, EY:, and EN: commands.

The formats for the JUMP IF YES and JUMP IF NO commands are:

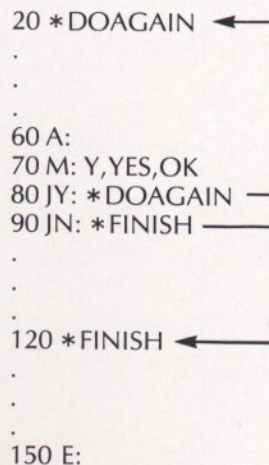
JY: *statement label*

JN: *statement label*

where: *statement label* is the destination point of the jump

JY: and *JN:* change the sequence of program execution upon a *yes* or a *no* match in the MATCH command. If a match is found and there is a JY: command, program execution will jump to the label specified in the JY: command (e.g., JY: *LOOP). If a match is **not** found and there is a JN: command, program execution will jump to the label specified in the JN: command (e.g., JN: *TIME). If both the JY: and JN: commands follow the same MATCH command, only one of the commands will be executed; the other command is bypassed.

Here is a fragment of a program demonstrating the function of the JY: and JN: commands:



Below is a sample program to demonstrate the M:, JY: and JN: commands:

```
10 R: *****
20 R: *   WHAT I SAW AT THE ZOO   *
30 R: *****
40 T: At the zoo there are LOTS of \
50 T: animals.
60 T:
70 T: The last time you were at the \
80 T: zoo, did you see any animals? \
90 A:
100 M: YES,Y
110 JN: *TRYAGAIN
120 *DOAGAIN
130 T:
140 T: What kinds of animals did you \
150 T: see?
160 T:
170 A:
180 T:
190 T: GREAT! Did you see anything \
200 T: else? \
210 A:
220 M: YES,Y
230 JY: *DOAGAIN
240 JN: *END
250 *TRYAGAIN
260 T:
270 T: Are you sure you didn't see \
280 T: any? \
290 A:
300 M: NO,N,NONE
310 JY: *DOAGAIN
320 T:
330 TN: OK, but I don't think you \
340 TN: looked very hard!
350 EN:
360 *END
370 T: You must have had a busy day at \
380 T: the zoo!
390 E:
```

As you can see, the program has three different statement labels: *DOAGAIN, *TRYAGAIN, *END. The program also uses both TY:, TN:, JY:, JN:, and EN: commands. A RUN of the program looks like this:

At the zoo there are LOTS of animals.

**The last time you were at the zoo,
did you see any animals? YES**

What kinds of animals did you see?

ELEPHANTS

GREAT! Did you see anything else? YES

What kinds of animals did you see?

LIONS

GREAT! Did you see anything else? YES

**.
.
.**

GREAT! Did you see anything else? NO

You must have had a busy day at the zoo!

READY

■

THE TRACE COMMAND

The TRACE command lets you follow the execution of your program step-by-step while the program is running. Often, when a program contains several JUMPs, it becomes difficult to follow the path of execution since the path no longer goes from lowest to highest line number. TRACE can help you to "debug" your programs (find and solve problems and mistakes) by letting you see exactly what your program is doing, each step of the way.

The TRACE command can be issued in either immediate or programmed mode.

The format for the TRACE command is:

TRACE: ON

TRACE: OFF

To demonstrate the TRACE command, issue the TRACE: ON command in immediate mode prior to execution of THE COLOR WHEEL program:

TRACE: ON

RUN

This is what you see as you execute THE COLOR WHEEL program:

```
--> 10 R:*****
--> 20 R:*   THE COLOR WHEEL   *
--> 30 R:*****
--> 40 T:THIS PROGRAM TESTS HOW
THIS PROGRAM TESTS HOW
--> 50 T:MANY COLORS YOU KNOW
MANY COLORS YOU KNOW
--> 60 T:

--> 70 T:ENTER A COLOR: \
ENTER A COLOR: --> 80 A:
RED
--> 90*DOAGAIN
--> 100 ,      T:

--> 110 ,      T:ENTER ANOTHER COLOR: \
ENTER ANOTHER COLOR: --> 120 ,      A:
VIOLET
--> 130 ,      J:*DOAGAIN
--> 90 *DOAGAIN
--> 100 ,      T:

--> 110 ,      T:ENTER ANOTHER COLOR: \
ENTER ANOTHER COLOR: --> 120 ,      A:

120 ,      A:
*** READY ***
■
```

The output at the bottom results from pressing the **BREAK** key to stop the program. But, we could TRACE the program forever if we wanted to.

■ When tracing long programs, the program trace will often scroll off the screen. To prevent this you can *freeze* and *unfreeze* the screen to stop the screen display. To *freeze* the screen, hold down the **CTRL** key and press the **1** key. The screen will instantly freeze and will not change until you unfreeze it. To *unfreeze* the screen, do the same thing: hold down the **CTRL** key and press the **1** key; the program trace will continue. ■

Once out of program execution (ie. program end, or pressing the **BREAK** key), to stop any further program trace, type: TRACE: OFF and press **RETURN**.

The TRACE command may also be inserted into a program by assigning line numbers to the TRACE: ON and TRACE: OFF commands. TRACE commands in a program allow you to trace the entire program or any portion of the program.

```
10 R:*****
20 R:*   THE COLOR WHEEL   *
30 R:*****
40 T: THIS PROGRAM TESTS HOW
50 T: MANY COLORS YOU KNOW
60 T:
70 T: ENTER A COLOR: \
80 A:
85 TRACE: ON
90 *DOAGAIN
```

```
100,      T:  
110,      T: ENTER ANOTHER COLOR: \  
120,      A:  
130,      J: *DOAGAIN  
140 E:
```

■ If the TRACE command is not “turned off” within the program, you must issue TRACE: **OFF** and press **RETURN** in immediate mode following program execution.

SECTION 5 SUMMARY

A *label* is a way of naming a PILOT statement line — a way to say it is special. The format for a statement label is:

line number **label name*

The label name must begin with a letter and cannot contain any embedded spaces or a space between the asterisk, *, and label name.

Examples:

```
15 *RIGHTON  
55 *ANDA123
```

The JUMP command tells the computer to break sequential program execution and jump to a specified label. The format for the JUMP command is:

J: **label name*

A JUMP in immediate mode starts program execution at the specified label. The JUMP command always jumps to the label with the lowest line number if two or more statements have the same label. A jump to a nonexistent label causes an error.

The JUMP IF YES and JUMP IF NO commands direct program execution to a specified label conditional upon the success of a match. Both commands may follow a single MATCH command, but they are mutually exclusive and only one will be executed per MATCH.

The formats for the JUMP IF YES and JUMP IF NO commands are:

JY: *label name*

JN: *label name*

The TRACE command allows you to follow the path of execution of a program step by step while it is running. The TRACE command is particularly useful in “debugging” programs.

The format for the TRACE command is:

TRACE: ON

TRACE: OFF

TRACE: ON and TRACE: OFF can be issued in immediate mode or programmed mode (with line numbers).

SECTION 5 QUIZ

1. A special statement is distinguished from other statements by assigning it a _____.
2. The special character identifying a label name is the _____.
 - (a) -
 - (b) *
 - (c) %
 - (d) \$
3. The JUMP command breaks sequential program execution and jumps to a specified _____.
4. If there are two statements with the same label, the JUMP command, J:, jumps to the label with the _____ line number.
5. A jump to a nonexistent label is ignored by PILOT. TRUE or FALSE?
6. The JY: and JN: commands _____ depending upon the success of a match.
7. The TRACE command displays the path of program execution after the program is terminated. TRUE or FALSE?
8. The TRACE command is useful for _____.
9. The TRACE command can be used in both immediate and program mode. TRUE or FALSE?

ANSWERS

1. Statement label, or label name, or label
2. (b)
3. Statement label, or label name, or label
4. Lower
5. FALSE
6. Direct program execution to a specified statement label
7. FALSE
8. Debugging a program
9. TRUE

ADVANCED BRANCHING TECHNIQUES

THE JUMP COMMAND IN IMMEDIATE MODE, J:

When the JUMP command is issued in immediate mode, it causes PILOT to automatically start program execution at the label name specified in the JUMP command. **WARNING:** Errors may result if you do not start program execution from the beginning.

THE JUMP ON MATCH COMMAND, JM:

The JUMP ON MATCH command is similar to the JUMP command, except that it jumps to one of several labeled statements, depending upon which MATCH string in the MATCH command produces a successful match.

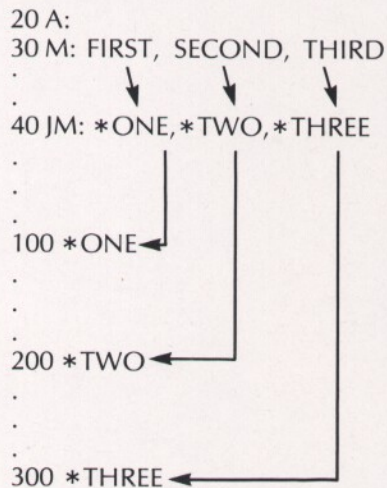
The format for the JUMP ON MATCH command is:

JM: label label label

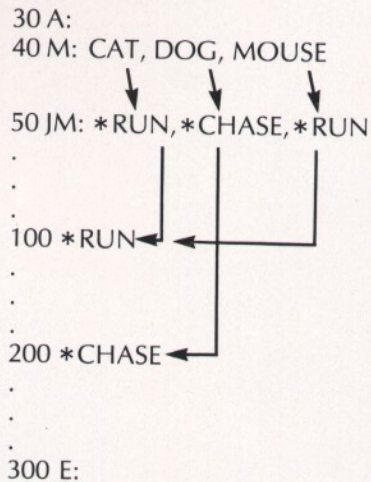
JM: label, label, label ...

The label names may be separated by either a space or a comma.

Each label corresponds to a string in the MATCH string. If the third match string produces a match, then the program jumps to the third label specified in the JM: command:



Below is a fragment of a program to further demonstrate how JM: works (all program statements unnecessary to the demonstration of this example have been omitted):



If the input string at line 30 is either CAT or MOUSE, the JM: command directs program execution to the label *RUN at line 100. If the input string is DOG, the JM: command directs program execution to the label *CHASE at line 200. If the input string is neither CAT, DOG, or MOUSE, the JM: command is bypassed.

MINI-QUIZ

1. The JUMP command may be issued in immediate mode to start program execution at the specified statement label. TRUE or FALSE?
2. If there is no match, the JM: command is _____.
3. In a JM: command, if the input string matches the fourth string of the MATCH command (M:), the program _____.
 - (a) Executes a JY: command
 - (b) Jumps to the first label specified in the JM: command
 - (c) Jumps to the first through fourth label specified in the JM: command.
 - (d) Jumps to the fourth label specified in the JM: command

ANSWERS

1. TRUE
2. Bypassed, skipped, or ignored
3. (d)

CONCEPTS INTRODUCED

- What Is a Variable?
- String Variables
 - What Is a String Variable?
 - Defining String Variables
 - Using TYPE and ACCEPT Commands With String Variables
 - What Are the Values of Your String Variables?
 - The DUMP Command
 - The COMPUTE Command, C:
 - String Concatenation
 - Growing Strings
 - Generating Plural Strings
 - The ACCEPT Command Without Input
 - The COMPUTE IF YES and COMPUTE IF NO Commands, CY:, CN:
 - Clearing Your Variables
- Section 6 Summary
- Section 6 Quiz
- Advanced Programming With Variables
 - Assigning Variables Into Other Variables
 - String Indirection
 - The MATCH STRING Command, MS:
 - Sensing a Match
 - Mini-Quiz

This section is about *variables*. Variables are an important part of PILOT programming. Beginners should read this section thoroughly. Advanced programmers already familiar with the concept of variables can read just the Section Summary and the Advanced Programming With Variables section.

WHAT IS A VARIABLE?

A *variable* is anything that is subject to change, like the time, date, or weather. In computer programming, a *variable* is anything whose value is changeable. Below are some examples:

$A = X * 5$

A and X are variables.

Today's date

is a variable. (It changes every day.)

Body weight

is a variable. (It changes all the time.)

A century

is NOT a variable. It is a fixed value of one hundred years.

Think of variables as compartments in the computer's memory:

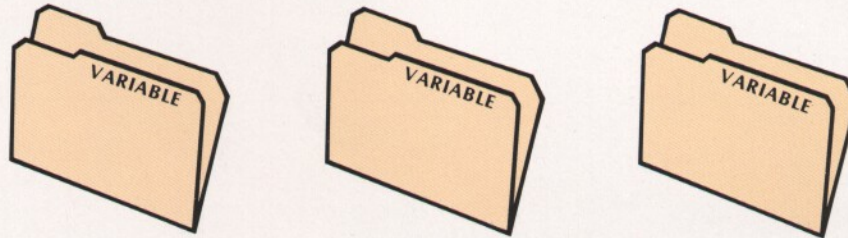


Figure 6-1 Concept of Variables

Because we may have more than one variable in memory, we must give each variable a *variable name* to distinguish it from all other variables.

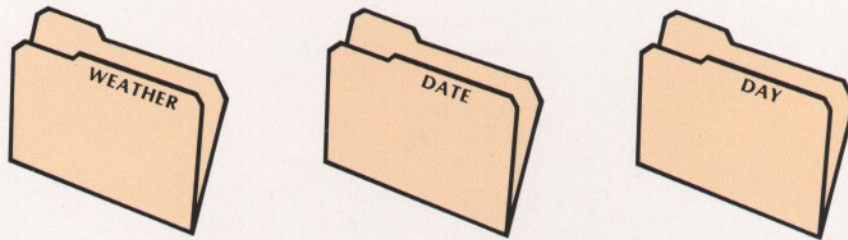


Figure 6-2 Assigning Variable Names

When assigning variable names, it is easier for you to identify a variable if each variable name represents the contents of the variable. The three variables above are good examples. Variable WEATHER holds values such as *sunny*, *cloudy*, or *rainy* that describe the weather. Variable DATE holds a *date*, (12/03/80). Variable DAY holds the day of the week, *Sunday*...*Saturday*. If we had named the variables 1,2,3 or A,B,C we might forget which variable holds what values. *Variable names*, just like the variables themselves, are *changeable*.

Let's look at our variables in memory.

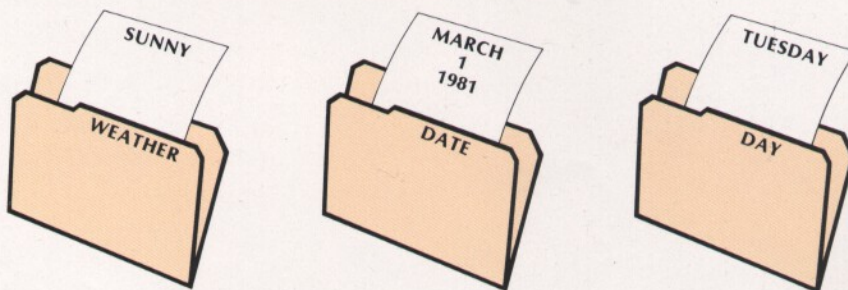


Figure 6-3 Assigning Values to Variables

Because variables have changeable values, the values may be changed at any time and any number of times. When a new value is assigned, the new value pushes the old value out; this is the concept of variables.

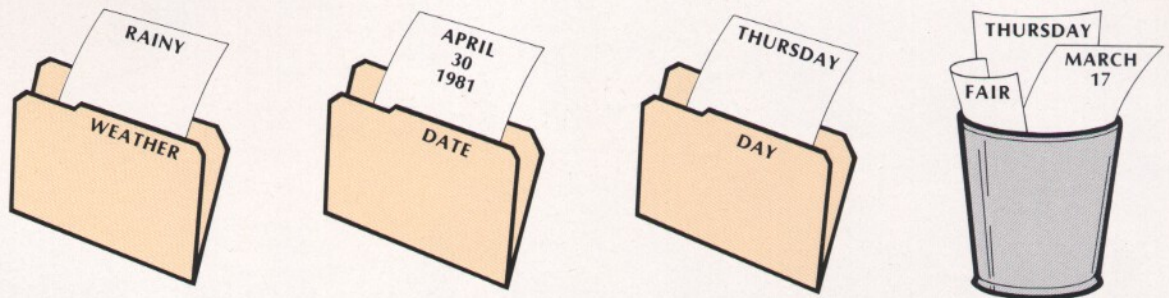


Figure 6-4 Changing a Variable's Value

In PILOT there are two types of variables: *string variables* and *numeric variables*. Numeric variables will not be discussed until Section 7, Using Numbers.

STRING VARIABLES

WHAT IS A STRING VARIABLE?

Do you remember the definition of a string? A string is any combination of letters, numbers, symbols, words, or phrases. A *string variable* is a type of variable whose value is a string.

String variables have string variable names. The variable name can be any combination of letters, numbers, or symbols up to 254 characters in length, preceded by a dollar sign, \$:

\$NAME	is a string variable name
\$DATE	is a string variable name
\$REPLY2ME	is a string variable name
\$12	is a string variable name
\$A	is a string variable name

DEFINING STRING VARIABLES

The computer's memory makes space (in the imaginary compartments for string variables at the time the string variable is defined. A *string variable* is defined when it is given a value. An undefined string variable has no value, like an empty compartment.

\$FARAWAY is an *undefined* string variable

An ACCEPT command (A:) can be used to define a string variable with keyboard input.

```
30 A: $FARAWAY
```

The keyboard input becomes the value of the string variable. If you enter CHINA in response to the above ACCEPT command, \$FARAWAY will be defined with a value of CHINA.

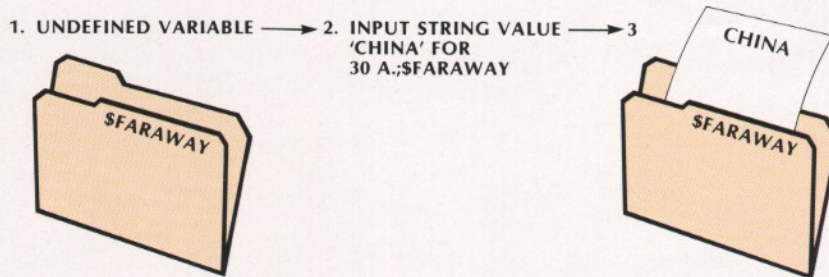


Figure 6-5 Defining a Variable With Input

USING TYPE AND ACCEPT COMMANDS WITH STRING VARIABLES

String variables are defined by the interaction of TYPE and ACCEPT commands. The ACCEPT command, A:, followed by a string variable name defines the string variable and saves the input as its value in memory.

```
10 T: WHAT IS YOUR NAME?
20 A: $NAME
30 T: WHERE DO YOU LIVE?
40 A: $ADDRESS
```

This is what happens when the above program is executed:

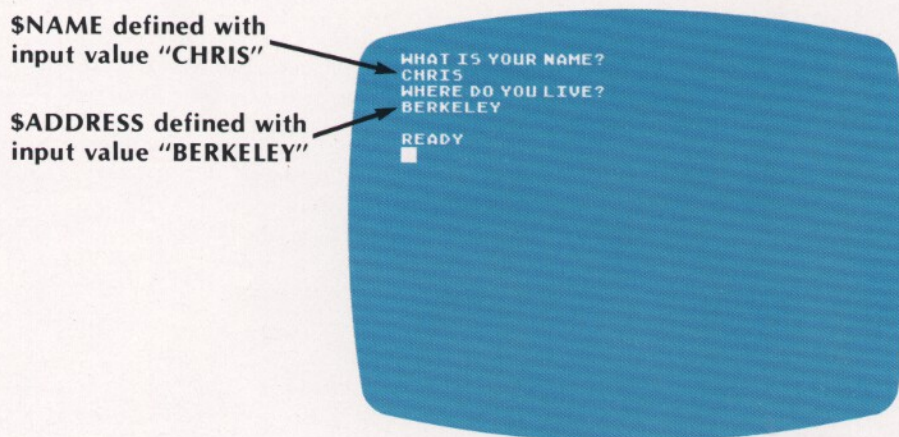


Figure 6-6 String Variables: Program Run

The TYPE command can also use string variables. If the string variable has been previously defined before the TYPE command, the string variable name in the TYPE statement is replaced with the value of the string variable. If it has not been defined, the string variable name is not replaced in the TYPE message. Type:

```
10 T: WHAT IS YOUR NAME?
20 A: $NAME
30 T: YOU, $NAME, ARE A VERY $FUNNY PERSON!
```

There are two string variables in the program: \$NAME and \$FUNNY. RUN the program. Enter your own name when prompted.

```
WHAT IS YOUR NAME?  
CHRIS  
YOU, CHRIS, ARE A VERY $FUNNY PERSON!
```

```
READY  
■
```

\$NAME is defined by your input to the ACCEPT command on line 20. In line 30 the string variable name \$NAME is replaced with your name. The string variable name \$FUNNY is not replaced because it was not defined before line 30. To fix the program to define \$FUNNY before line 30, insert the following lines:

```
23 T: WHAT KIND OF PERSON ARE YOU?  
26 A: $FUNNY
```

RUN the program again.

```
WHAT IS YOUR NAME?  
CHRIS  
WHAT KIND OF PERSON ARE YOU?  
NICE  
YOU, CHRIS, ARE A VERY NICE PERSON!
```

```
READY  
■
```

See how easy it is to use variables?

Following is an Automatic Letter Writer. You can use this program to create a letter choosing the date, the city you live in, who the letter is addressed to, and how you're feeling when you write the letter.

```
10 T: AUTOMATIC LETTER WRITER  
20 T:  
30 T: (please answer the following questions)  
40 T:  
50 T: What is your name? \  
60 A: $NAME  
70 T: Where do you live, $NAME?  
80 T: in \  
90 A: $CITY  
100 T: What is today's date? \  
110 A: $DATE  
120 T: Who do you want to send this letter to?  
130 T: (complete the following)  
140 T: Dear \  
150 A: $TO  
160 T: (Now type one to three sentences about  
170 T: out how you're feeling and what you  
180 T: are doing. Remember the periods!)  
190 T: (press RETURN after each sentence or  
200 T: if you are finished with sentences)  
210 T: > \  
220 A: $IMDOING1  
230 T: > \  
240 A: $IMDOING2
```

```
250 T: > \           █ SENTENCE PROMPT
260 A: $IMDOING3
270 T: (press RETURN to see the letter)
280 A:
290 T:
300 T:                $DATE
310 T:                $CITY
320 T:
330 T:
340 T: Dear $TO,
350 T:
360 T:                How are you doing? Fine I\
370 T: hope. $IMDOING1 $IMDOING2 $IMDOING3
380 T:                Anyway, I just wanted you to know\
390 T: I was thinking of you. Bye now.
400 T:
410 T:                Yours,
420 T:
430 T:
440 T:
450 T:
460 T: P.S. Please write soon.
```

WHAT ARE THE VALUES OF YOUR STRING VARIABLES?

The DUMP Command

After you've executed a program using string variables, you can find the values of the string variables by typing the executive command, DUMP. As the command label implies, DUMP *dumps* or displays the string variable names and their values on the screen.

A DUMP command following the execution of our example program might display something like this:

```
READY
DUMP
$NAME = 'CHRIS'
$FUNNY = 'NICE'
```

■ The value of each string variable is enclosed in single quotation marks. Notice that 'CHRIS' and 'NICE' are the values entered in the example program. ■

The COMPUTE Command C:

The COMPUTE command assigns strings into string variables.

To assign a string into a string variable, use the following format:

C: *variable name = string*

where: *variable name* is the variable name receiving the string
string is the assigned value

■ The equals sign separates the variable name from the assigned string. ■

Examples:

C: \$BLOOP= CAT assigns the string 'CAT' into the variable \$BLOOP

C: \$NUM= ONE assigns the string 'ONE' into the variable \$NUM

C: \$ADDR= 1234 GEARY ST. assigns the string '1234 GEARY ST.' into the variable \$ADDR

C: \$AGE= 10 assigns the string '10' into the variable \$AGE

String Concatenation

The COMPUTE command also enables you to put strings together. This is called *concatenation*.

Growing Strings

Following is an example of string concatenation. Type in the following two program statements:

```
10 C: $NAME =  
20 C: $STAR = *
```

RUN the program. You will see only a READY message and the cursor. Now, enter DUMP and press **RETURN**. You should see:

```
READY  
DUMP  
$NAME= ''  
$STAR= '*'
```

\$NAME="" means that the value of \$NAME is an empty string, whereas \$STAR has the value '*'.
\$STAR= '*'

Enter the remainder of the program:

```
30 *GROW C: $NAME = $NAME$STAR  
40 , T: $NAME
```

LIST and RUN the program:

```
LIST  
10 C: $NAME =  
20 C: $STAR = *  
30 *GROW C: $NAME = $NAME$STAR  
40 , T: $NAME  
READY  
RUN
```

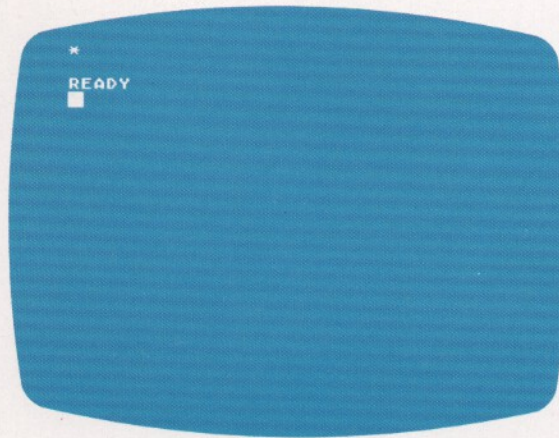


Figure 6-7 Growing Strings: Program List and Run

Issue a DUMP command again:

```
READY
DUMP
$NAME='*'
$STAR='*'
READY
■
```

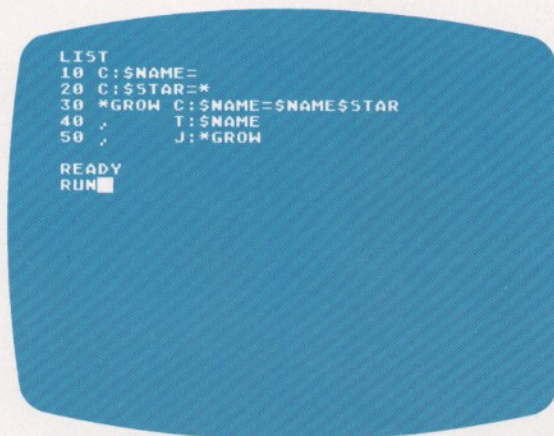
■ Look at the value of \$NAME now! Line 30 concatenated the value of \$STAR onto the end of the value of \$NAME. In effect, this added a '*' onto the end of an empty string, making the value of \$NAME a '*'. ■

Notice that there are no spaces between the two variable names \$NAME\$STAR. ■

Add this line to the end of the program:

```
50 J: *GROW
```

LIST and RUN the program:



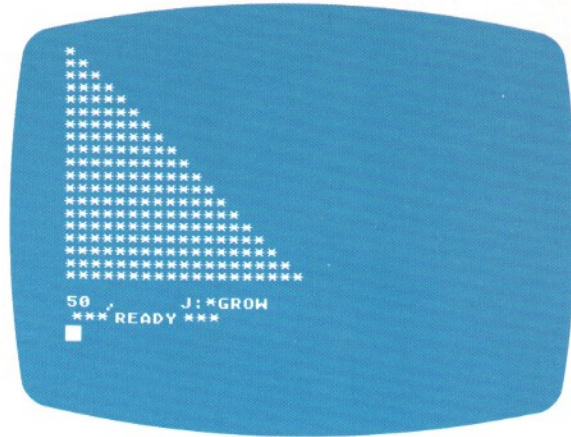


Figure 6-8 Growing Strings: With a Second DUMP Command

Look at how \$NAME keeps growing!

Press the **BREAK** key, and issue the DUMP command again:

```
DUMP
$NAME= '*****'
$STAR= '*'
READY
■
```

\$NAME seems to have grown somewhat!

Generating Plural Strings

Concatenating strings allows you to generate plurals; to change a singular string into a plural string. Clear memory with the NEW command and type in this program:

```
10 T:WHAT IS YOUR FAVORITE FLOWER?
20 A:$FLOWER
30 T:WOULD YOU LIKE A DOZEN $FLOWER?
40 A:
```

RUN the program:

```
WHAT IS YOUR FAVORITE FLOWER?
ROSE
WOULD YOU LIKE A DOZEN ROSE?
YES

READY
■
```

A DOZEN ROSE doesn't sound right, and it certainly is not proper English. We need to change the input string ROSE to ROSES before it is displayed at line 30. To do this we must add two lines and change line 30. Type in:

```
24 C: $$ = S
28 C: $FLOWERS = $FLOWER$$
30 T: WOULD YOU LIKE A DOZEN $FLOWERS?
```

Line 28 assigns the string variable \$FLOWERS the value of the "concatenation" of \$\$ to the end of \$FLOWER. In line 30, the variable name \$FLOWER is changed to \$FLOWERS.

■ In line 28, notice that there are no spaces between the two variable names \$FLOWER\$\$\$. Also, notice that there are no spaces on either side of the equals sign in lines 24 and 25. ■

LIST the program:

```
10 T:WHAT IS YOUR FAVORITE FLOWER?
20 A:$FLOWER
24 C:$S=S
28 C:$FLOWERS=$FLOWER$$
30 T:WOULD YOU LIKE A DOZEN $FLOWER$$?
40 A:
```

Now, RUN the program:

```
WHAT IS YOUR FAVORITE FLOWER?
ROSE
WOULD YOU LIKE A DOZEN ROSES?
YES

READY
■
```

Perfect!

THE ACCEPT COMMAND WITHOUT INPUT

The **A: =** form of the ACCEPT command accepts the value of a variable instead of input from the keyboard.

The format of ACCEPT without input is:

A: = *variable name*

where: *variable name* is the name of the variable whose value is placed in the accept buffer.

Examples:

A: = \$FRIEND	places the value of \$FRIEND into the accept buffer
A: = \$VAR	places the value of \$VAR into the accept buffer
A: = \$K	places the value of \$K into the accept buffer

Enter the program below:

```
10 C:$NAME =
20 C:$STAR = *
30 *GROW C: $NAME = $NAME$STAR
40,      T: $NAME
50,      A: = $NAME
60,      M: *****
70,      JN: *GROW
```

Line 30 concatenates an asterisk onto the end of the value of \$NAME. Line 50 places the value of \$NAME into the accept buffer. Line 60 searches for the value of \$NAME in the accept buffer for a match. If there are less than seven asterisks attached to the value of \$NAME, the program jumps up to *GROW. When seven asterisks are added to \$NAME, the program stops.

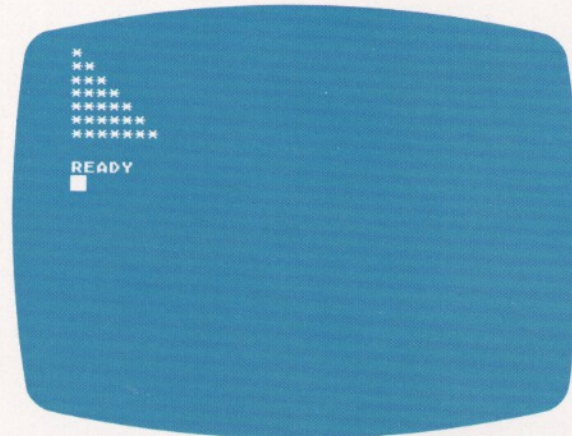


Figure 6-9 ACCEPT Without Input: Program Run

Using the A: = form of the ACCEPT command can save you from repeatedly inputting the same value.

THE COMPUTE IF YES AND COMPUTE IF NO COMMANDS, CY:, CN:

The COMPUTE command may be used with yes or no conditioners to make COMPUTE conditional upon a match or no match.

The formats for the COMPUTE IF YES and COMPUTE IF NO commands are:

CY: *variable name=value*

CN: *variable name=value*

A MATCH command must precede the CY: or CN: command. If the match is successful, the CY: command is executed (e.g., CY: \$VAR=CAT). If the match is unsuccessful, the CN: command is executed (CN: \$VAR=DOG). If both commands follow the same MATCH command, only one will be executed. The other command is bypassed.

■ The CY: and CN: commands are applicable to computing numeric strings, which is discussed in Section 7. ■

CLEARING YOUR VARIABLES

The NEW VARIABLES command, VNEW:, allows you to clear string variables within a program. Clearing a string variable means to *undefine* the string variable. The NEW VARIABLES command can be issued in either immediate or program mode.

The format for the NEW VARIABLES command is:

VNEW:\$ clear all string variables only

SECTION 6 SUMMARY

A *variable* is anything that is subject to change. PILOT has two types of variables: string and numeric.

String variables store strings. A string is a series of letters, numbers, symbols, words, phrases, or a combination thereof. A *string variable name* is any combination of letters or numbers, preceded by a dollar sign '\$'.

The TYPE and ACCEPT commands work together to let you input values to string variables. The TYPE message tells the user what type of value to input. The ACCEPT command followed by a string variable name saves the input in memory. Variables are defined when a value is entered with the ACCEPT command or assigned by the COMPUTE command. The COMPUTE command assigns strings, numbers, or variable names to variables.

The format for the COMPUTE command is:

C: *variable name* = *value*

Singular strings can be made plural through string *concatenation*. Concatenation means to join two strings together. By joining a variable whose value is 'S' to the end of a variable with a singular value, the singular variable becomes plural.

C: \$ROSE = ROSE

C: \$S = S

C: \$ROSES = \$ROSE\$S

Values of the variables can be displayed by issuing the DUMP command in immediate mode. The variable's values will be displayed in this format:

string variable name = '*value*'

The VNEW command issued in immediate or programmed mode, allows you to clear, or "undefine" all string variables within a program. The format for the VNEW command is:

VNEW: \$

SECTION 6 QUIZ

1. PILOT uses two types of variables. They are _____ and _____.
2. _____ is an example of a string variable name.
 - (a) #DATE
 - (b) %ANT
 - (c) #Z
 - (d) \$X
3. A string variable can have a number as a value. TRUE or FALSE?
4. The COMPUTE command (C:) assigns: _____.
 - (a) Strings into variables
 - (b) Names to variables
 - (c) a and b
5. A variable is defined when it is assigned a value by the COMPUTE command or when _____.
6. The executive command that displays the values of string variables is the _____ command.
7. The A: = form of the ACCEPT command _____ instead of input from the keyboard.
8. Concatenation is the joining together of two strings, end to end. To make the value of string variable \$ANIMAL plural, fill in the missing program statements:
10 T: WHAT IS YOUR FAVORITE ANIMAL?
20 A: \$ANIMAL
30 C:
40 C:
50 T: DO YOU SEE MANY \$ANIMAL\$S IN THE ZOO?
60 A:

ANSWERS

1. String variables, numeric variables
2. (d)
3. TRUE
4. (a)
5. A value is input with the ACCEPT command.
6. DUMP
7. Assigns the value of a variable to the accept buffer.
A: = \$NAME
8. 30 C: \$S=S
40 C: \$ANIMALS=\$ANIMAL\$S

ADVANCED PROGRAMMING WITH VARIABLES

ASSIGNING VARIABLES INTO OTHER VARIABLES

The value of variables can be assigned into other variables with the COMPUTE command in the format:

C: $\$$ *variable name* = *variable name*

Examples:

C: \$HERMAN = \$GEORGE

C: \$NUM = \$ONE

C: \$GEO = \$BOX

STRING INDIRECTION

Values of string variables can themselves be other string variable names. The dollar sign (\$) is both a string variable indicator and an indirection operator.

String indirection can best be explained by example:

C: \$LADDER = JANE assigns \$LADDER the value 'JANE'

C: \$JANE = ATARI assigns \$JANE the value 'ATARI'

C: \$ATARI = LUNCH assigns \$ATARI the value 'LUNCH'

T: \$LADDER produces 'JANE'

T: \$\$LADDER produces 'ATARI'

T: \$\$\$LADDER produces 'LUNCH'

T: \$\$\$LADDER produces '\$LUNCH'

As many \$'s as desired may precede the variable name portion. String indirection is allowed anywhere a simple string variable is allowed.

THE MATCH STRING COMMAND, MS:

The MATCH STRING command acts exactly like the MATCH command, except that it produces three pre-named string variables upon a successful match.

The format for the MATCH STRING command is:

MS: *string*

MS: *string, string...*

MS: *variable name*

where: *string* can be any character, number, word, symbol, phrase, or a combination of these.

Like MATCH, if MATCH STRING specifies several strings, the strings must be separated by commas or vertical bars.

If MATCH STRING specifies a variable name, the value of the variable is the match string.

If there is a match, MS: *divides the input into three pre-named string variables: \$LEFT, \$MATCH, and \$RIGHT:*

\$LEFT is the value of everything to the *left* of the matched item
\$MATCH is the value of the matched item
\$RIGHT is the value of everything to the *right* of the matched item

This example program demonstrates how the MATCH STRING command divides input into three variables:

```
10 T: ENTER A STRING: \  
20 A:  
30 MS: AND  
40 T: THE LEFT VARIABLE IS: $LEFT  
50 T: THE MATCH VARIABLE IS: $MATCH  
60 T: THE RIGHT VARIABLE IS: $RIGHT
```

The string that will produce a match is AND. The input string I LIKE CATS AND DOGS produces this output:

```
ENTER A STRING: I LIKE CATS AND DOGS  
THE LEFT VARIABLE IS: I LIKE CATS  
THE MATCH VARIABLE IS: AND  
THE RIGHT VARIABLE IS: DOGS
```

```
READY  
■
```

■ Remember, that a string variable's value replaces its variable name in a TYPE statement (if defined). ■

If a match is unsuccessful, the three strings (\$LEFT, \$MATCH, and \$RIGHT) retain any value they had prior to the MATCH STRING command.

The \$LEFT or \$RIGHT variables may have a null, or no value, depending upon where in the input string the match occurred as shown below: (A *null value* means that its value is nothing; this is different than undefined.)

```
ENTER A STRING: AND  
THE LEFT VARIABLE IS:  
THE MATCH VARIABLE IS: AND  
THE RIGHT VARIABLE IS:
```

```
READY  
■
```

The ability to separate a single string into three separate strings makes MATCH STRING a very powerful command. You can use it to rearrange your string input, discard portions of your input, and so on.

SENSING A MATCH

When the ACCEPT and MATCH strings are evaluated for a successful or unsuccessful match, the condition is remembered as true or false in the computer's memory. You can display the MATCH SENSING variable to find out if the match is true or not.

The MATCH SENSING variable %M contains the value of 1 if the match is true, and a value of 0 if the match is false.

Example:

T: %M
0 (0 means no match)

T: %M
1 (1 means a match)

The %M variable may be displayed in either immediate or programmed mode with the TYPE command. It is a very useful variable for debugging (finding mistakes) in your programs.

MINI-QUIZ

1. The value of variables can be assigned into other variables with the _____ command.
 - (a) A:
 - (b) C:
 - (c) A: =
 - (d) M:
2. The dollar sign (\$) is both a string variable indicator and an indirection operator. TRUE or FALSE?
3. The MATCH STRING command acts exactly like the MATCH command, except that it divides the input string into three string variables. They are: _____
 - (a) \$BEGIN, \$MIDDLE, \$END
 - (b) \$LEFT, \$MATCH, \$RIGHT
 - (c) \$LEFT, \$MIDDLE, \$RIGHT
 - (d) \$FIRST, \$SECOND, \$THIRD
 - (e) \$L, \$M, \$R

ANSWERS

1. (b)
2. TRUE
3. (b)

USING NUMBERS

CONCEPTS INTRODUCED

Maximum Number Size
Modulo
Numeric Types
 Constants
 Numeric Variables
 Assigning Value to Numeric Variables
 The COMPUTE Command, C:
 The Random Number
Arithmetic Operators
Operator Precedence
Relational Operators
Section 7 Summary
Section 7 Quiz

To come this far, you have mastered many new concepts in your quest to learn PILOT. The next concept is one you are already familiar with: *numbers*!

The world of PILOT graphics and sound revolves around numbers. There are a few key rules and ideas to understand in order to use numbers; these are the topic of this section.

MAXIMUM NUMBER SIZE

The largest positive number PILOT understands is 32767. The smallest negative number PILOT understands is -32768. PILOT handles arithmetic in strange ways if you go beyond the +32767 and -32768 limits. For example, the equation:

$32767 + 1$ will return -32768 instead of 32768.

PILOT arithmetic uses only *integers* in the -32768 to 32767 range. *Fractional numbers are not allowed*. The remainder from the division of two integers is represented by a special arithmetic operator, called *modulo*.

MODULO

The symbol for modulo is the backslash character, \, located on the \ . + key. *Modulo represents the remainder of one integer divided by another.*

The *modulo* of one number by another number is the remainder of the division of the two numbers.

$a \text{ number} \backslash \text{another number} = \text{remainder of } (a \text{ number} / (\text{another number}))$

This is how modulo works:

PILOT division functions like longhand division. First, you divide the dividend by the divisor:

$$\begin{array}{r} \text{DIVISOR} \longrightarrow 3 \overline{) 7} \\ \underline{-6} \\ 1 \end{array} \begin{array}{l} \longleftarrow \text{QUOTIENT} \\ \longleftarrow \text{DIVIDEND} \\ \longleftarrow \text{REMAINDER} \end{array}$$

Figure 7-1 Division Problem

Dividing 7 by 3 equals 2 with a remainder of 1. PILOT keeps the quotient (the answer) and drops off the remainder.

$$7/3=2$$

The remainder of 7/3 must be calculated by using the modulo operator, `\`, as shown in the following example:

$$7\backslash 3=1$$

The remainder of 7/3 = modulo of 7\3 = 1

$$\begin{array}{r} 3 \overline{) 7} \\ \underline{-6} \\ 1 \end{array} \begin{array}{l} \longleftarrow \text{QUOTIENT} \\ \longleftarrow \text{MODULUS} \end{array}$$

Figure 7-2 The Modulo of 7\3

Here are more examples:

$$5 \div 4 = 5/4 = 1 \text{ (quotient)}$$

$$5 \backslash 4 = 1 \text{ (remainder)}$$

$$999 \div 12 = 99/12 = 8 \text{ (quotient)}$$

$$99 \backslash 12 = 3 \text{ (remainder)}$$

$$10 \div 5 = 10/5 = 2 \text{ (quotient)}$$

$$10 \backslash 5 = 0 \text{ (remainder)}$$

NUMERIC TYPES

There are two types of numbers in PILOT: *constants* and *variables*.

CONSTANTS

Constant numbers never change. A 4 will always remain a 4. It will never be a 5 or a 2/3. It is "constant."

NUMERIC VARIABLES

Variables change. A *numeric variable* has a changeable value. A numeric variable is a variable whose changeable value is *always* a number.

Numeric variables have names. They are represented by the number sign (#) followed by a single letter ranging from A to Z:

#J #X #O #K #A

#X=#Y-#Z #X,#Y, and #Z are numeric variables.

#A=4+6 #A is a numeric variable; 4 and 6 are constants.

Numeric variables have names. They are represented by the number sign, #, followed by a single letter ranging from A to Z:

#J #X #O #K #A

■ An important fact to remember is that because there are only 26 letters in the alphabet, there are only 26 available numeric variable names. ■

ASSIGNING VALUE TO NUMERIC VARIABLES

The same rules that govern string variables also apply for defining numeric variables. One difference is that numeric variables are automatically assigned a value of 0 instead of "undefined."

Numeric variables can acquire an input value from the ACCEPT command.

A: # *numeric variable name*

Examples:

A: #N

A: #X

Numeric variables can also be assigned value with the COMPUTE command described below.

THE COMPUTE COMMAND, C:

The COMPUTE command (C:) evaluates a numeric expression and assigns its value to a numeric variable.

The format for COMPUTE is:

C: *numeric variable* = *numeric expression*

Examples:

C: #X=5*4

C: #A=-X/4

C: #B=10

All numeric calculations must be done with the COMPUTE command.

Here is an example program:

```
10 T:THIS PROGRAM ADDS TWO NUMBERS
20 T:
30 T: ENTER A NUMBER: \
40 A: #A
60 T: ENTER ANOTHER NUMBER: \
70 A: #B
80 T:
90 C: #Z=#A+#B
100 T: THE SUM IS #Z
```

In this program, two numbers are entered with the ACCEPT command and assigned to numeric variables. Line 90 calculates the sum of #A and #B and assigns the sum to #Z. Line 100 prints the answer. A RUN of the program looks like this:

```
THIS PROGRAM ADDS TWO NUMBERS

ENTER A NUMBER: 10
ENTER ANOTHER NUMBER: 5

THE SUM IS 15

READY
■
```

This example program uses the COMPUTE command to assign the product of two variables into a numeric variable:

```
10 T: THIS IS A MULTIPLICATION PROGRAM
20 T: ENTER TWO NUMBERS: \
30 T:
40 T: FIRST NUMBER: \
50 A: #A
70 T: SECOND NUMBER: \
80 A: #B
90 C: #X=#A*#B
100 T:
110 T#A x #B = #X
120 E:
```

The COMPUTE statement on line 90 calculates the product of #A and #B, and assigns the product into numeric variable #X. Execute the program with RUN and press **RETURN**.

```
THIS IS A MULTIPLICATION PROGRAM
ENTER TWO NUMBERS:

FIRST NUMBER: 20
SECOND NUMBER: 6

20 x 6 = 120

READY
■
```

See how the COMPUTE command assigns value to variables?

THE RANDOM NUMBER

A *random number* is any integer number generated by the computer between -32768 and +32767. The question mark (?) in a numeric expression generates a random number. The *modulo operator* (\) limits the range of the random number selected. Study the examples below:

?	generates a random number between -32768 and 32767 inclusive.
?\4	generates a random number between 0 and 3 inclusive.
?\8	generates a random number between 0 and 7 inclusive.
?\10	generates a random number between 0 and 9 inclusive.

When you limit the selected random number with the modulo operator, the result will always be either a 0 or a positive integer less than the limiting number.

ARITHMETIC OPERATORS

ATARI PILOT uses four symbols to designate addition, subtraction, multiplication, and division:

+	addition
-	subtraction
*	multiplication
/	division

OPERATOR PRECEDENCE

Unlike many arithmetic systems, PILOT has no special order in which the arithmetic operators (+, -, *, /) are executed. In PILOT, a numeric expression is evaluated from the left to the right:

$$4 + 4/2 \text{ equals } 8/2 \text{ equals } 4$$

■ To change the order of precedence, put a set of parentheses around the part of the expression that you want evaluated first. Thus, in the above example, if you want the 4/2 to be evaluated first instead of the 4 + 4, place the parentheses around the 4/2:

$$4 + (4/2) \text{ equals } 4 + (2) \text{ equals } 6$$

As you can see, the order of precedence can change the value of the expression.

Numeric expressions are not limited to only one set of parentheses: rather, two sets of parentheses can be *nested* inside of one another. The expression enclosed by the innermost set of parentheses is evaluated first.

$$5 + ((2*6)/12) \text{ equals } 5 + ((12)/12) \text{ equals } 5 + (1) \text{ equals } 6 \quad \blacksquare$$

RELATIONAL OPERATORS

Relational operators are a rather advanced concept. Relational operators compare two numbers or numeric expressions and determine if one is greater than, less than, equal to, or not equal to the other:

#A=#B	#A is equal to #B
#A<>#B	#A is not equal to #B
#A<#B	#A is less than #B
#A>#B	#A is greater than #B
#A<=#B	#A is less than or equal to #B
#A>=#B	#A is greater than or equal to #B

The number or numeric expression to the left of the relational operator is compared to the number or numeric expression to the right of the relational operator.

A relational operator *declares* whether the comparison is true or false.

5 > 4	is TRUE
5 > 6	is FALSE
0 = 0	is TRUE
0 < = -37	is FALSE
#C = 0	is TRUE or FALSE depending upon the value of #C
#A > #B	is TRUE or FALSE depending upon the value of #A and #B

Relational operators yield as follows:

- A value of 1 if a declaration is true and a value of 0 if it is false — just like a match and no match.
- The declaration $5 \geq 2$ is true and this yields a value of 1; the declaration $5 < 3$ is false and yields a value of 0.

Relational operators can make program statements conditional by enclosing them in parentheses and placing them between the command label and the colon in any statement, as follows:

command label (relational expression) action of the instruction

Examples:

T (#C > 5): VERY GOOD JOB!

A (#N < > #R):

J (# > = #B): *DOAGAIN

■ The relational expression must be enclosed within parentheses. The relational expression must be placed between the command label and the colon, ∴ ■

If a statement contains a relational expression, it is executed only if the expression is true.

The following example demonstrates the use of relational, or conditional, expressions.

```
10 C: #T = 200
20 T: THIS PROGRAM WILL RUN \
30 *NOTAGAIN
40,   T: FOREVER AND \
50,   C: #T = #T - 1
60,   J (#T > 0): *NOTAGAIN
70,   T: EVER! WHEW!!
80,   E:
```

The program counts down from 200 to 0. As long as the value of #T is greater than 0, program execution jumps to the *NOTAGAIN label. Once #T = 0, the conditional relational expression #T > 0 is false and the JUMP command is not executed.

Run the program and let it stop by itself. Below is a "shortened" RUN of the program:



Figure 7-3 FOREVER Program Run

SECTION 7 SUMMARY

PILOT uses integer numbers in the range of -32768 to $+32767$.

Fractional numbers are represented by a special operator called *modulo*. Modulo, represented by the backslash character (`\`) is the resultant remainder from a division of two numbers.

PILOT has two types of numbers: constants and variables. *Constants* don't change value; *numeric variables* do. Numeric variable names are represented by a number sign, #, followed by a single letter from A through Z.

```
#A #M #Z
```

Numeric variables can have only numeric values.

PILOT generates *random numbers* with a question mark (?) in a numeric expression. The backslash character (\) limits the random number generated.

```

?           generates a number between -32768 and 32767
?\3        generates 0, 1, or 2
?\5        generates 0, 1, 2, 3, or 4
  
```

The COMPUTE command evaluates a numeric expression and assigns its value to a numeric variable. The format for COMPUTE is:

C: *numeric variable* = *numeric expression*

```
C: #A = 5 + 5
```

```
C: #Z = #A / 4
```

PILOT uses four arithmetic operators. They are:

```

+           addition
-           subtraction
*           multiplication
/           division
  
```

PILOT evaluates numeric expressions from left to right. Parentheses placed around any segment of an expression change the order of *precedence*. Whatever is enclosed by the parentheses will be evaluated first, If parentheses are nested, the innermost set is evaluated first.

Relational operators are used to compare two numbers or numeric expressions. When evaluated, a relation will either be true (producing a value of 1) or false (producing a value of 0). The relational operators are:

=	#A=#B	#A is equal to #B
<>	#A<>#B	#A is not equal to B
<	#A<#B	#A is less than #B
>	#A>#B	#A is greater than #B
<=	#A<=#B	#A is less than or equal to #B
>=	#A>=#B	#A is greater than or equal to #B

Relational expressions can make a program statement conditional. A program statement is conditionally executed only if its relational expression evaluates to true.

Examples:

T(#A>#B): ENTER ANOTHER NUMBER:

A(#N<>#R):

J(#V>=0): *LOOP

The relational expression, enclosed within parentheses, must follow the command label and precede the colon.

SECTION 7
QUIZ

1. PILOT allows _____ numbers in the range from _____
_____ to _____.
2. To express fractional numbers, the _____ operator, represented by the _____ is used.
3. $11\sqrt{=}$ _____.
4. What are constants? _____.
5. What are numeric variables? _____.
6. Numeric variable names are shown in _____ below:
 - (a) \$A
 - (b) #A
 - (c) %Z
 - (d) #NUM
 - (e) \$NUM
 - (f) b and d
 - (g) a and e
7. A random number is generated by a _____ anywhere in a numeric expression.
8. What is the function of the COMPUTE command, C:, when used with numeric variables? _____.
9. An equals sign (=) must always appear in the COMPUTE command. TRUE or FALSE?
10. Match the arithmetic operators:

-	addition
/	subtraction
+	multiplication
*	division
11. _____ change the order of precedence in a numeric expression.
12. Relational operators compare two numeric expressions to decide if one expression is equal to, greater or less than, greater or equal to, or less than or equal to the other. TRUE or FALSE?
13. Which conditional TYPE command is correct?
 - (a) T: (#C>#A) THAT'S RIGHT!
 - (b) T (#C>#A):THAT'S RIGHT!
 - (c) T: #C>#A: THAT'S RIGHT!

ANSWERS

1. Integer, -32768, + 32767
2. Modulo, backslash, \
3. 4. The remainder of $(11 \setminus 7) = 4$
4. Constants are numbers that never change their value.
5. Numeric variables are numbers with changeable values.
6. (b)
7. Question mark, (?)
8. The COMPUTE command evaluates numeric expressions and assigns the value to a numeric variable.
9. TRUE
10.

-	—	addition
/	—	subtraction
+	—	multiplication
*	—	division
11. Parentheses
12. TRUE
13. (b)

CONCEPTS INTRODUCED

What Is a Module?
 The Parts of a Module
 Calling a Module
 The USE Command, U:
 The Module End
 The END Command, E:
 Nested Modules
 Conditional Modules
 The USE IF YES and USE IF NO Commands, UY:, UN:
 Example: The Calculator Program
 Section 8 Summary
 Section 8 Quiz

WHAT IS A MODULE?

As you become a more experienced PILOT programmer and start writing longer and more complicated programs, you will find that many program statements are often used repeatedly within the same program. You could repeat the statements each time they are needed, but this can become lengthy and tedious. Instead, group the statements to be used more than once, and branch to them as they are needed. *This group of reusable statements is called a **module**.* Modules are like miniature programs within a big program.

Modules are a powerful technique to streamline your PILOT programs. With modules you can:

- Make large PILOT programs easier to write, modify, and understand
- Conveniently save and reuse sections of programs
- Make PILOT programs shorter

A program with several modules may look like this:

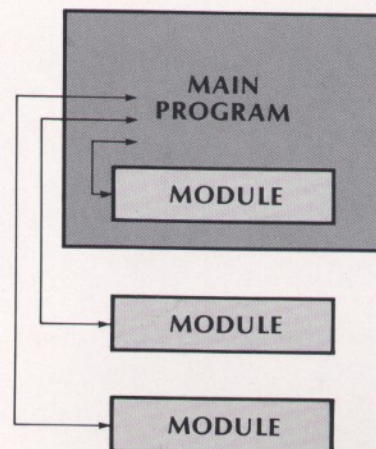


Figure 8-1 Program With Several Modules

THE PARTS OF A MODULE

A module is: *reusable*

A module has: *a label*

a task, e.g., getting input or displaying messages.

an end. The END command (E:) placed at the last line of the module serves as an exit and returns program execution back to the statement following the call to the module.

A module has the form shown in Figure 8-2.

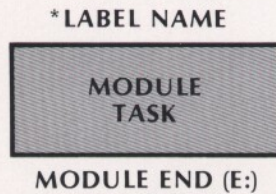


Figure 8-2 Module Structure

Here is a sample module:

```
500 *GETNAME
510 ,      T: ENTER A NAME: \
520 ,      A: $NAME
530 ,      E:
```

*GETNAME is the label of the module. Lines 510-520 perform the function of the module. The END command at line 530 is the module's end.

The above module, *GETNAME, could be branched to whenever a name needs to be entered in the program. At the end of the module, program execution branches back into the main program.

CALLING A MODULE

The USE Command, U:

The USE command *calls, or branches to a module.* It tells the computer to call a module into use.

The format for the USE command is:

U: *module label*

where: *module label* is a label at the beginning of a module.

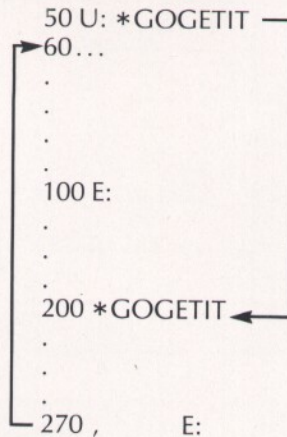
Examples:

```
50 U: *INFO
75 U: *START
```

■ The USE command is always part of the main program. It is *not* part of a module. ■

When the USE command tells the computer to branch to a module at a specified label, it is telling the computer to jump to the line containing that label. The difference between a USE and a JUMP command is that when the USE command is executed, the computer remembers the line number of the statement immediately following the USE. When the module is finished, program execution automatically continues with the line number following the USE command.

This is how it works:



■ Modules can be placed following the program END command, E:, because program execution will jump around the END command to branch to a module. However, it's usually best to place modules at the beginning of a program because the computer can find them faster. ■

THE MODULE END

The END Command, E:

■ Every module must end with the END command. ■

The format for the END command is:

E:

Section 3 stated that the END command ends the program. The same END command used in a module ends that module, and signals to the computer to return to the main program. END within a module does *not* end the entire program. Your ATARI computer is smart: it knows the difference between a module END and a program END.

NESTED MODULES

Modules can be nested. *Nesting means that a module can call another module, which in turn can call a third module, and so forth.* PILOT allows up to eight nested modules in one program. There is nothing new to worry about when nesting modules. PILOT remembers the correct line number to return to after each module exit.

Here is a diagram of what a program with five nested modules could look like:

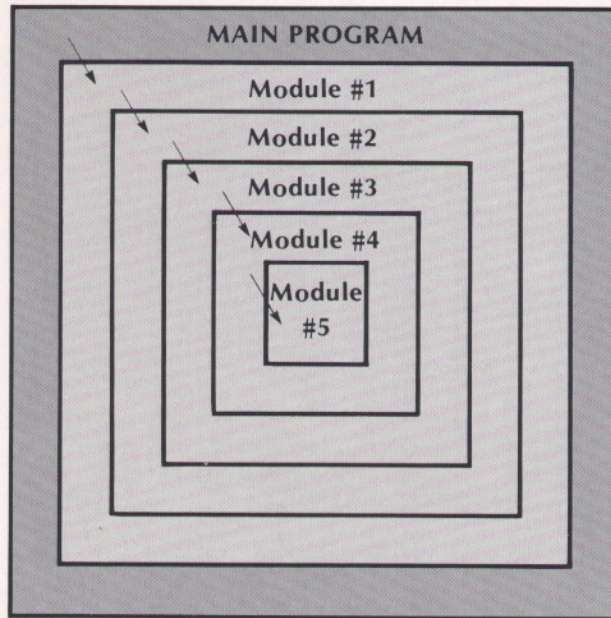


Figure 8-3 Nested Modules

It is a good idea to debug programs like this with the TRACE: ON and TRACE: OFF commands.

CONDITIONAL MODULES

THE USE IF YES AND USE IF NO COMMANDS UY:, UN:

In Section 4 you learned that the TYPE, JUMP, and END commands can be called conditionally, depending upon the success of a match between an ACCEPT and MATCH string. Modules can be called conditionally also by changing the USE command to a USE IF YES or USE IF NO command.

These commands have the format:

UY: *module label*

UN: *module label*

Depending upon a match or no match between the ACCEPT and MATCH string, one of these commands will be executed. If both are present, one will be bypassed.

Here is part of a program that uses the UY: and UN: commands:

```
10 T: DO YOU WANT TO PLAY A GAME?  
20 A:  
30 M: Y,YES  
40 UY: *GAME  
50 UN: *WORDS  
60 E:
```

```

100 *GAME
.
.
500 E:
600 *WORDS
.
.
900 E:

```

If a "y" or "yes" response is entered, the program plays the game by calling the *GAME module. If the answer is not a match, the program calls the *WORDS module.

EXAMPLE: THE CALCULATOR PROGRAM

Following is a rather complex program. It is called CALCULATOR because it will calculate the answer to most simple addition, subtraction, multiplication, and division problems. This program demonstrates not only modules and nested modules, but also many decision-making commands such as MATCH, MATCH STRING, JUMP ON MATCH, and END IF NO commands. It's a lengthy program, it's but well worth the effort to type it in and see how it works.

```

10 *START
20 U: *GETPROBLEM
30 U: *COMPUTE
40 U: *SHOWRESULT
50 T: ANOTHER EQUATION? \
60 A:
70 M: Y,YES,YEAH,OK
90 JY: *START
100 E:
200 *GETPROBLEM
210 ,      T: PLEASE ENTER A MATH PROBLEM: \
220 ,      A: $PROBLEM
230 ,      T:
240 ,      E:
300 *COMPUTE
310 ,      U: *SEPARATE
320 ,      A: = $PROBLEM
330 ,      M: +,*,-,/
340 ,      JM: *ADD,*MULT,*SUB,*DIV
350 ,      E:
400 *SHOWRESULT
410 ,      T: $PROBLEM = $RESULT
420 ,      E:
500 *SEPARATE
510 ,      MS: +,*,-,/
520 ,      EN:
530 ,      A: #I = $LEFT
540 ,      A: #J = $RIGHT
550 ,      E:
600 *ADD
610 ,      C: #R = #I + #J
620 ,      C: $RESULT = #R
630 ,      E:

```

```
640 *MULT
650 ,      C: #R = #I * #J
660 ,      C: $RESULT = #R
670 ,      E:
680 *SUB
690 ,      C: #R = #I - #J
700 ,      C: $RESULT = #R
710 ,      E:
720 *DIV
730 ,      C: #R = #I / #J
740 ,      C: $RESULT = #R
750 ,      C: #Q = #I \ #J [get remainder]
760 ,      C (#Q > 0): $RESULT = #R rem #Q
770 ,      E:
```

LIST and RUN the program. A sample RUN should look something like this:

```
PLEASE ENTER A MATH PROBLEM: 5+7
5+7 = 12
ANOTHER ONE? Y
PLEASE ENTER A MATH PROBLEM: 100/20
100/20 = 5
ANOTHER ONE? N
READY
■
```

SECTION 8 SUMMARY

A *module* is a group of repeatedly used statements clustered together. They are building blocks which can make programs shorter and easier to understand. Modules are powerful tools because they are reusable.

Every module must have a *label*, a *task*, and an *end*. When the module is needed, the main program branches to the module. When the module has been executed, the module branches back to the main program.

The USE command, located in the main program, *calls* the module into action. The format for the USE command is:

U: *module label*

Examples:

U: *MINI

10 U: *COMPUTE

100 U: *K555

Every module must end with an END command. The format for the END command is:

E:

The END command, when executed within a module, ends only the module and NOT the program. END transfers program execution to the statement following the calling USE command in the main program.

Modules can be nested within one another, allowing a module to call another module. Up to eight modules can be nested in one PILOT program.

The USE command can be used conditionally to call modules, depending upon the success of an ACCEPT and MATCH string match. The formats for the USE IF YES and USE IF NO commands are:

UY: *statement label*

UN: *statement label*

The conditional command rules apply for the USE IF YES and USE IF NO commands.

SECTION 8 QUIZ

1. A group of repeatedly used statements, separated from the main flow of a program is called a _____.
2. Modules must have a:
 - (a) Label, USE command, END command
 - (b) Label, task, END command
 - (c) Label, task, end
 - (d) USE command, task, END command
3. A module label is like any other label name. TRUE or FALSE?
4. The USE command must be in the first statement in a module. TRUE or FALSE?
5. The USE command specifies the _____ of the module to be called.
 - (a) Line number
 - (b) Label
6. The END command of a module has no effect on the main program. The E: ends only the module. TRUE or FALSE?
7. What is a nested module? _____.
8. Up to _____ modules may be nested in one program.
 - (a) 2
 - (b) 4
 - (c) 6
 - (d) 8
 - (e) 10
9. Modules can be called conditionally with the _____ or _____ commands.

ANSWERS

1. Module
2. (b)
3. TRUE
4. FALSE
5. (b)
6. TRUE
7. A module called by another module
8. (d)
9. USE IF YES command, UY:, USE IF NO command, UN:



MANY VOICES, MANY SOUNDS

CONCEPTS INTRODUCED

Definition List
 The SOUND Command, SO:
 Immediate Mode Sounds
 Program Mode Sounds
 The PAUSE Command, PA:
 Random Music
 A Special Song
 Section 9 Summary
 Section 9 Quiz
 Advanced Programming With Sound
 The PAUSE Command: Increments of 1/60ths

PILOT has a special command that lets you create sound and musical tones. It is called the SOUND command. In Section 2 you were briefly introduced to the SOUND command when you learned to play a C scale and a chromatic scale. This section will show you how to program the SOUND command so that you can write music.

DEFINITION LIST

Below is a list of definitions you will need to make sounds:

tone A tone is a musical note from low C (one and a half octaves below middle C) to high F# (one and a half octaves above middle C). Each of these tones has a corresponding number from 0 to 31.

chord Three or more tones played at the same time.

voice A voice represents one tone in a musical chord. The SOUND command can be used to play up to four voices at the same time.

MIDDLE C So called because in music this note is in the middle of the music staff.

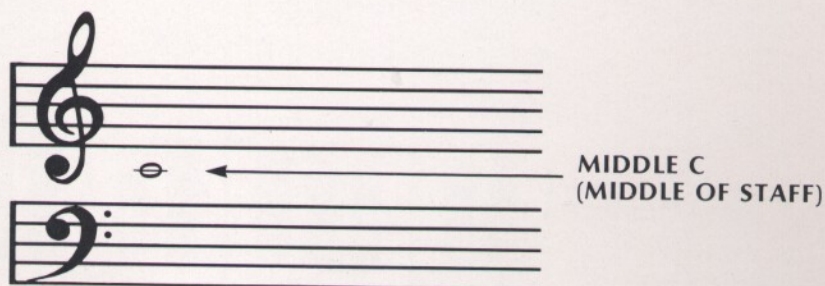


Figure 9-1 Position of Middle C

HALF STEP The smallest interval between two tones.

WHOLE STEP An interval of two half steps between two tones.

MAJOR SCALE A musical pattern consisting of the following steps: whole, whole, half, whole, whole, whole, half (see Figure 9-3).

OCTAVE An interval of eight notes ranging from one note to the same note higher on the musical scale, e.g., middle C to the C above middle C is one octave. For example, C, D, E, F, G, A, B, and C.

THE SOUND COMMAND, SO:

The format for the SOUND command is:

SO: voice

SO: voice voice ...

SO: voice, voice, voice, voice

■ Each voice must be separated by a space or a comma. Up to four voices can follow a SOUND command. ■

Examples:

SO: 1

SO: 1,5,8

SO: 0

SO: 0 means no sound, or OFF.

Below is a diagram showing the correlation between tones and their numerical values. The diagram shows the musical scale and a piano keyboard like the one shown in Section 2.

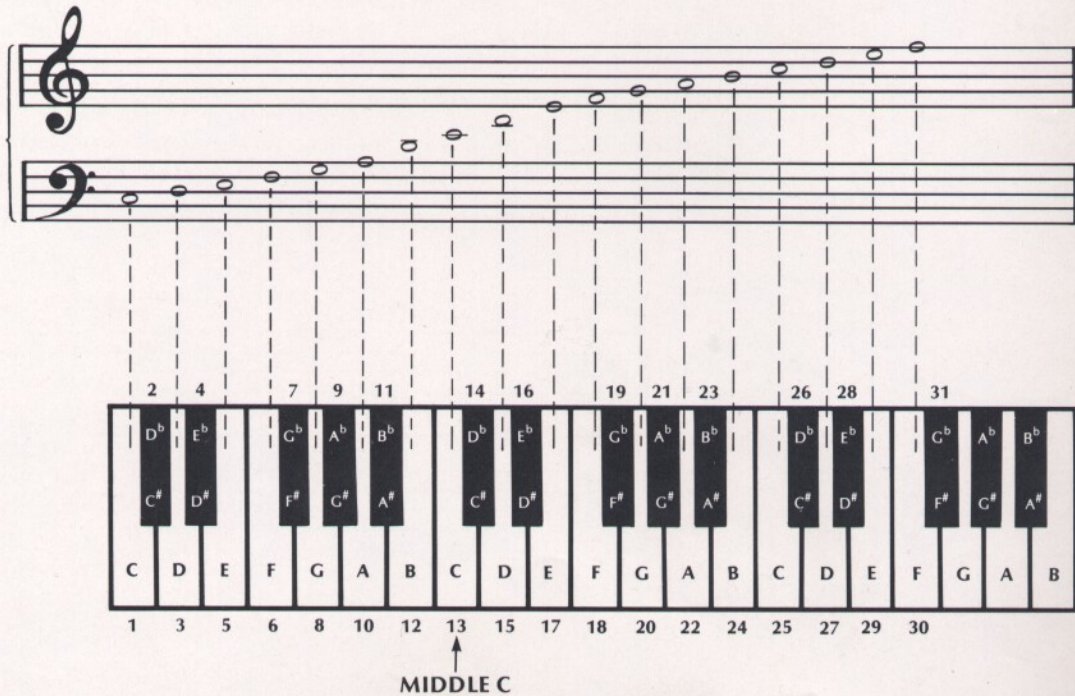


Figure 9-2 Relation of Piano Keyboard to Musical Scale

The following diagram shows several musical scales:

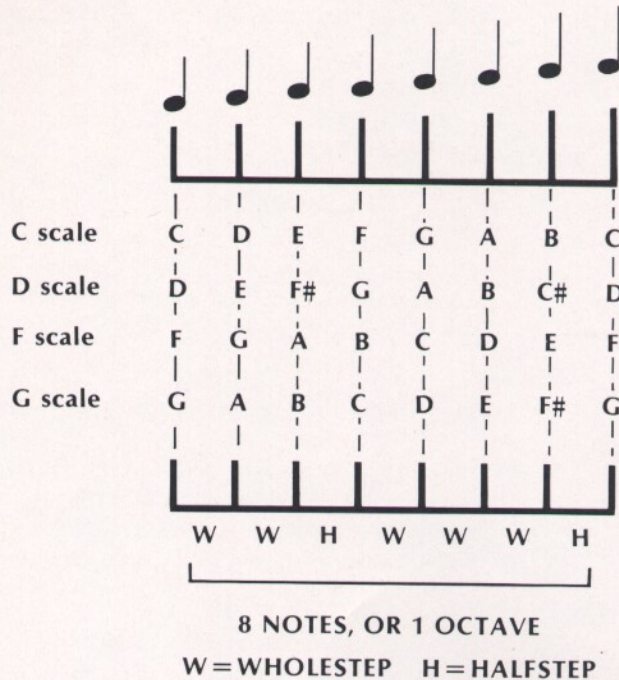


Figure 9-3 Musical Scales

Keep these diagrams as a handy reference when using the SOUND command. When you determine what note, or tone, you want to play (C,D,F#,A...), look up the note and its corresponding numerical value on one of the charts and plug it into the SOUND command.

You can use the SOUND command in both immediate and program mode.

Immediate Mode Sounds

Section 2 explained how to use the SOUND command in immediate mode. Like all other immediate mode commands as soon as you press the **RETURN** key, the computer responds. In this case, the computer responds by generating a tone. The sound continues to play until you either type another SOUND command, stop the sound with a SO: 0 command, or press the **BREAK** key. However, you can't make sounds in rapid succession unless you are a very fast typist!

Enter the following SOUND commands: (make sure the volume is turned up on your television set).

SO: 13 **RETURN**
SO: 15 **RETURN**
SO: 13 **RETURN**
SO: 14 **RETURN**

Depending upon how fast you can type, it may take a long time to type in each tone. To speed up the tone changes, put the SOUND commands in a program.

Program Mode Sounds

Program mode allows you to play a series of SOUND commands in rapid succession once the program is RUN. Type:

10 SO: 13
20 SO: 15
30 SO: 13
40 SO: 14
50 E:

RUN the program. The program sounds like a short beep. The program executes so fast that you can't hear each individual tone. The PAUSE command used *in conjunction with* the SOUND command helps to extend the length of tones so you can hear them.

THE PAUSE COMMAND, PA:

The PAUSE command tells the computer to pause, or delay program execution temporarily. PAUSE does not affect the sound in progress; the sound continues playing while the computer pauses before executing the next statement. The length of the pause determines the length of the tone.

The format for the PAUSE command is:

PA: numeric expression

Examples:

PA: 2

PA: #K *5/2

PA: 32

Below is a chart showing you what PAUSE values to use to determine the length of each note. The chart is based on a "Moderato" tempo.







Note:						
	Whole	Half	Quarter	Eighth	Sixteenth	Thirty-second
Tempo Andante	128	64	32	16	8	4

Figure 9-4 PAUSE Values for Moderato (Andante) Tempo

These values are not absolute. If you want to change the tempo faster or slower, change the PAUSE values. All PAUSE values are relative to each other. If you change the PAUSE value of a quarter note from 16 to 64 to change the tempo, then you must change the PAUSE value of the whole, half, eighth, and sixteenth notes as well. (The PAUSE values are calculated by powers of two: 2, 4, 8, 16, 32, 64, 128, 256...)

Examine this diagram:

Note:	Whole	Half	Quarter	Eighth	Sixteenth	Thirty-second	
Largo	512	256	128	64	32	16	(very slow)
Adagio	256	128	64	32	16	8	
Andante	128	64	32	16	8	4	
Allegro	64	32	16	8	4	2	
Presto	32	16	8	4	2	1	(very fast)

Figure 9-5 PAUSE Values for Different Tempos

You can change the example program to play each tone more slowly by adding a PAUSE command following each SOUND command. To extend each tone, insert a PA: command after each SOUND command:

```

10 SO: 13
20 PA: 16
30 SO: 15
40 PA: 16
50 SO: 13
60 PA: 16
70 SO: 14
80 PA: 16
90 E:

```

RUN the program. Each tone should be succinct and clear.

RANDOM MUSIC

Random music is generated by computing random numerical values for SOUND and PAUSE commands with the random operator, ?.

The following program generates random music:

```

10 C: #A= ?
20 C: #B= ?
30 *START
40 , C(?): #A=#A+ ?
50 , C(?\3-1): #B=#B+ ?
60 , SO: #A,#B
70 , PA: ?\32
80 , J: *START

```

Lines 10 and 20 compute two random numbers for numeric variables #A and #B. Lines 40 and 50 recompute the values to two different random numbers. At line, 60 SO: #A, #B, the first voice plays #A. The second voice plays #B. The PAUSE command at line 70 generates a random length of a pause between 0 and 31, varying the length of the tone. The JUMP at line 80 jumps to the label at line 30 to begin the loop again. To stop this program, press the **BREAK** key.

A SPECIAL SONG

The following program plays a tune to show you what can be done with the SOUND command. Type it in and give it a RUN.

```
1 *START
5 C: #T = #T + 1
10 SO: 17,13,8
20 PA: 24
30 SO: 15,13,8
40 PA: 8
50 SO: 13,8
60 PA: 32
70 SO: 15,13,8
80 PA: 32
90 SO: 17,13,8
100 PA: 32
110 SO: 17,13,8
120 PA: 32
130 SO: 17,13,8
140 PA: 64
150 SO: 15,12,8
160 PA: 32
170 SO: 15,12,8
180 PA: 32
190 SO: 15,12,8
200 PA: 64
210 SO: 17,13,8
220 PA: 32
230 SO: 20,17,13,8
240 PA: 32
250 SO: 20,17,13,8
260 PA: 64
270 SO: 17,13,8
280 PA: 24
290 SO: 15,13,8
300 PA: 8
310 SO: 13,8
320 PA: 32
330 SO: 15,13,8
340 PA: 32
350 SO: 17,13,8
360 PA: 32
370 SO: 17,13,8
380 PA: 32
390 SO: 17,13,8
400 PA: 32
410 SO: 15,12,8
420 PA: 32
430 SO: 15,12,8
440 PA: 32
450 SO: 17,13,8
460 PA: 32
470 SO: 15,12,8
480 PA: 32
490 SO: 13,8,5,1
500 PA: 192
510 SO: 0
520 J(#T < 2): *START
```

SECTION 9 SUMMARY

The SOUND command lets you play musical tones on your ATARI computer. The SOUND command generates tones.

A *tone* is a musical note from low C to high F#. Each tone has a corresponding number from 0 to 31.

A *voice* represents one tone in a musical chord. Up to four voices may be played at once in a single SOUND command.

A *half step* is the smallest interval between two tones.

A *whole step* is an interval of two half steps between two tones.

A *major scale* is a musical pattern consisting of the following steps: whole, whole, half, whole, whole, whole, half (see Figure 9-3).

An *octave* is an interval of 8 notes in a major scale beginning with any note and ranging to the same note 12 half steps higher on the musical scale.

The format for the SOUND command is:

SO: *voice*

SO: *voice voice ...*

SO: *voice,voice ...*

Each voice in a SOUND command must be separated by a space or a comma.

When SOUND commands are played in immediate mode the tone lasts until the next SOUND command is issued, or the **BREAK** key is pressed.

The SOUND commands in programmed mode play very rapidly unless used in conjunction with PAUSE commands. PAUSE commands always follow SOUND commands.

The PAUSE command tells the computer to delay program execution temporarily. Everything is delayed except the SOUND command. The PAUSE command has the format:

PA: *number or numeric expression*

Because PAUSE doesn't affect the SOUND command, it is effective for determining the length of tones. The length of the pause determines the length of the preceding tone. Example:

SO: 15

PA: 16

For convenience, the whole, half, quarter, eighth, and sixteenth notes for various tempos have been assigned a PAUSE value. These values can be referred to in Figure 9-5.

PILOT generates *random music* with the random number operator (?). Used within a numeric expression in the SOUND and/or PAUSE command, ? generates random tones or pauses.

SO: ?\5*6

PA: ?\5+20

SECTION 9 QUIZ

1. The SOUND command generates musical _____.
2. A total of _____ tones can be played at one time.
3. Tones must be played in the range of _____.
 - (a) 0-99
 - (b) 0-30
 - (c) 0-31 where 0= OFF
 - (d) 1-31 where 1= OFF
4. Each voice in a SOUND command must be separated by either a _____
_____ or a _____.
5. Match the following:

half step	interval of two half steps
whole step	half steps played sequentially
octave	interval of a tone
chromatic scale	8 notes in a major scale
6. The PAUSE command tells the computer to _____
_____.
7. The PAUSE command must precede the SOUND command. TRUE or FALSE?
8. To generate random music, the _____ is used to compute
the SOUND and PAUSE values.

ANSWERS

1. tones
2. four
3. (c)
4. space, comma
5. half step — interval of two half steps
whole step — half steps played sequentially
octave — interval of a tone
chromatic scale — 8 notes is a major scale
6. Delay program execution for a specified length of time.
7. FALSE (PAUSE follows the SOUND command.)
8. Random number operator, ?.

ADVANCED PROGRAMMING WITH SOUND

THE PAUSE COMMAND: INCREMENTS OF 1/60ths

In more technical terms, the PAUSE command delays program execution for 60ths of a second. Each increment of one indicates a 1/60th of a second delay. The PAUSE command may be used to synchronize musical and graphical presentations. PAUSE may also be used any place in a program where a delay in execution is required or desirable.

Examples:

PA: 20 (will pause $20/60 = 1/3$ of a second)

PA: #K/5*2 (will pause $(\#K/5*2)/60 = \#K/150$ seconds)

PA: 3600 (will pause $3600/60 = 60$ seconds)

The following chart should help you understand how to determine the length of a PAUSE in 1/60ths of a second:

PAUSE VALUE	LENGTH
1	1/60 second
2	1/30 second
6	1/10 second
10	1/6 second
15	1/4 second
20	1/3 second
30	1/2 second
60	1 second
120	2 seconds
3600	1 minute

Figure 9-6 PAUSE Values in 60ths of a Second

WELCOME TO GRAPHICS

CONCEPTS INTRODUCED

- The GRAPHICS Command, GR:
- The Graphics Screen
 - Cartesian Coordinates
 - Turtle Geometry
- GRAPHICS Subcommands
 - The PEN Subcommand
 - Subcommands Using Cartesian Coordinates
 - The GOTO Subcommand
 - The DRAWTO Subcommand
 - The FILLTO Subcommand
 - Subcommands Using Turtle Geometry
 - The TURNTO Subcommand
 - The TURN Subcommand
 - The DRAW Subcommand
 - The GO Subcommand
 - The FILL Subcommand
- Repeating GRAPHICS Commands
- Home, Sweet Home Program
- Section 10 Summary
- Section 10 Quiz
- Advanced Programming With GRAPHICS
 - GRAPHICS Variables
 - %X
 - %Y
 - %A
 - %Z
- Mini-Quiz

Congratulations! You've made it to ATARI PILOT graphics, one of the most enjoyable and flexible features of your computer and the PILOT programming language. This section enables you to enter the world of figures, pictures, and color.

PILOT cannot draw anything unless it is in *graphics mode*. In graphics mode, the screen looks like this:



Figure 10-1 The GRAPHICS Screen

This is the graphics screen — all black except for a small strip of blue at the bottom of the screen.

THE GRAPHICS COMMAND, GR:

The GRAPHICS command tells the computer to enter graphics mode and open the graphics screen.

The format for the GRAPHICS command is:

GR: *subcommand*

Because there are several different *types* of graphics capabilities, they can't all be handled under one command without the help of *subcommands*. The graphic subcommands tell the computer what to do in graphics mode.

Below are the subcommands available with the GRAPHICS command, GR:

CLEAR
DRAW
DRAWTO
FILL
FILLTO
GO
GOTO
PEN
QUIT
TURN
TURNTO

Each subcommand is described in detail in this section.

THE GRAPHICS SCREEN

Type: **GR: CLEAR** and press **RETURN** to display the graphics screen. (The cursor remains in the same position.) As you see, it is quite different from the blue text screen we've been using so far. Four lines of text can be displayed at a time in the text window at the bottom of the screen.

PILOT supports two drawing systems simultaneously when in graphics mode. They are the:

- Cartesian coordinates (an absolute reference system)
- "Turtle" geometry (a relative polar reference system in which the origin is always carried with the moving point — the turtle)

CARTESIAN COORDINATES

Think of your graphics screen as having an invisible grid laid over it. The grid is made up of 12,800 units: 160 units across and 80 units high. Actually, the complete graphics screen extends far beyond the borders of the visible computer screen: up to 32767 units in two directions and to -32768 units in the other two directions.

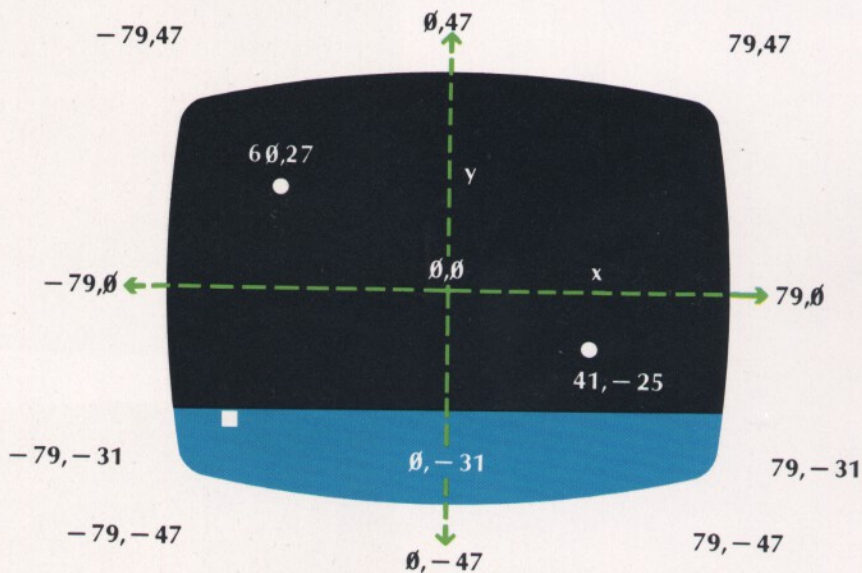


Figure 10-2 Cartesian Coordinates

The screen is also divided into four quadrants, divided by the X (horizontal) axis and the Y (vertical) axis. The X and Y axes intersect in a point at the center of the screen. This point, called the *center point* or *origin* has the coordinates 0,0. Every point within each quadrant can be located and identified by a pair of coordinates in the form (X,Y). A *coordinate* defines a point's position relative to the origin. The first number is the X coordinate: the distance along the X axis from the point to the origin. (The value of X increases to the right of the origin, and decreases to the left.) The second number is the Y coordinate: the distance along the Y axis from the point to the origin. (The value of Y increases upward from the origin and decreases downward away from the origin.) This is labeled in the diagram.

For example, point A (-60,27) in the diagram is 60 units to the left of the origin, and 27 units upward from the origin. Point B (41, -25) is 41 units to the right of the origin and 25 units below the origin.

The subcommands that use the Cartesian coordinates are:

DRAWTO
FILLTO
GOTO

TURTLE GEOMETRY

Turtle geometry is a graphics system developed by Dr. Seymour Papert and the LOGO group at the Massachusetts Institute of Technology. In turtle geometry, the graphics display screen becomes the playfield of a tiny invisible turtle. The turtle is always at the "center pole" of an invisible circle that moves with him wherever he goes. This circle is divided into 360 one degree angles with 0 and 360 degrees pointing toward the top of the screen. No matter where the turtle is located, it will always be pointing in a directional angle measured in degrees, called **THETA**.

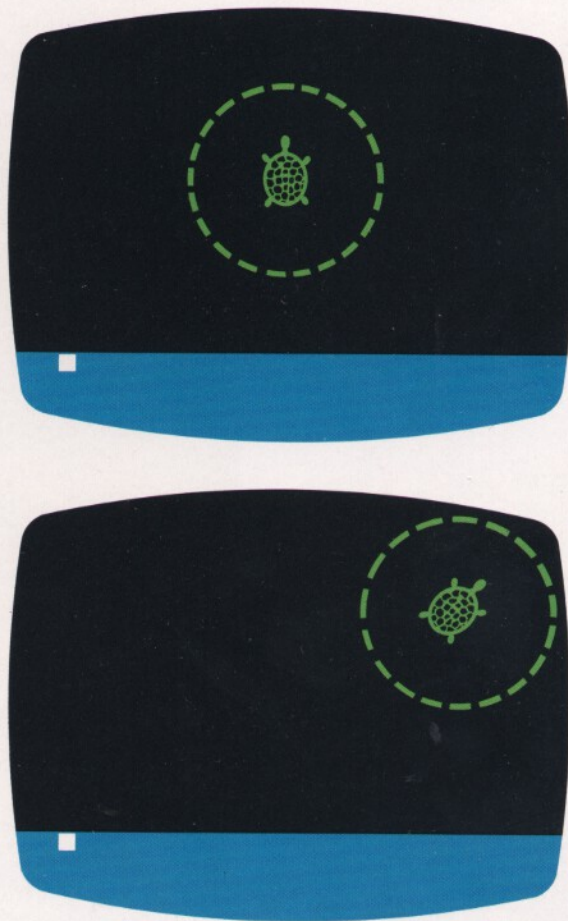


Figure 10-3 The GRAPHICS Turtle

If the turtle walks straight forward from the origin (center of the screen) to the top of the screen, its path is the 0 degree or 360 degree angle (the beginning and ending of the circle). As the turtle moves on the screen, he carries this reference circle with him. The turtle is always at the center of the circle.

END CIRCLE AT THETA = 360° BEGIN CIRCLE AT THETA = 0°

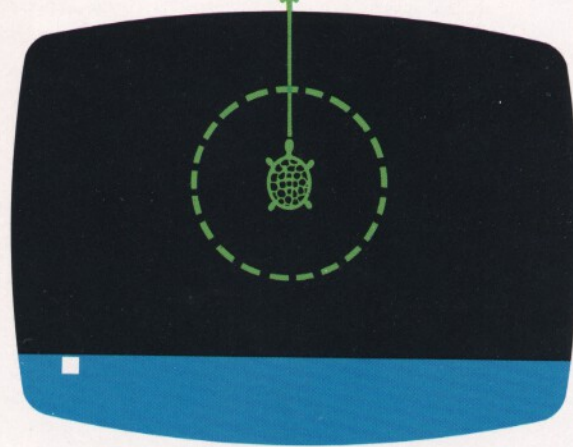


Figure 10-4 The 0- and 360-Degree Angle

The turtle can turn to the left or right in 360 different angles. The turtle pointing directly to the right (a quarter turn of the circle), is facing 90 degrees from the 0 degree angle. The turtle facing the bottom of the screen (a half turn of the circle), is pointing 180 degrees from the 0 degree angle. Turning the turtle to the right until it points directly to the left side of the screen faces the turtle towards the 270 degree angle. Of course, these aren't the only directions that the turtle can face: it can be positioned to face toward any angle between 0 and 360 degrees.

The turtle can also be turned to the left; however, calculating its direction is a little more complicated. To determine its angle, you must count counterclockwise from 0 to 360 degrees, and then add a negative sign to the degree value. Thus, if the turtle is facing the lower left corner, you calculate its angle by counting counterclockwise from 0. The degree value is 135. Add a negative sign to the front of 135, and the degree angle is -135.

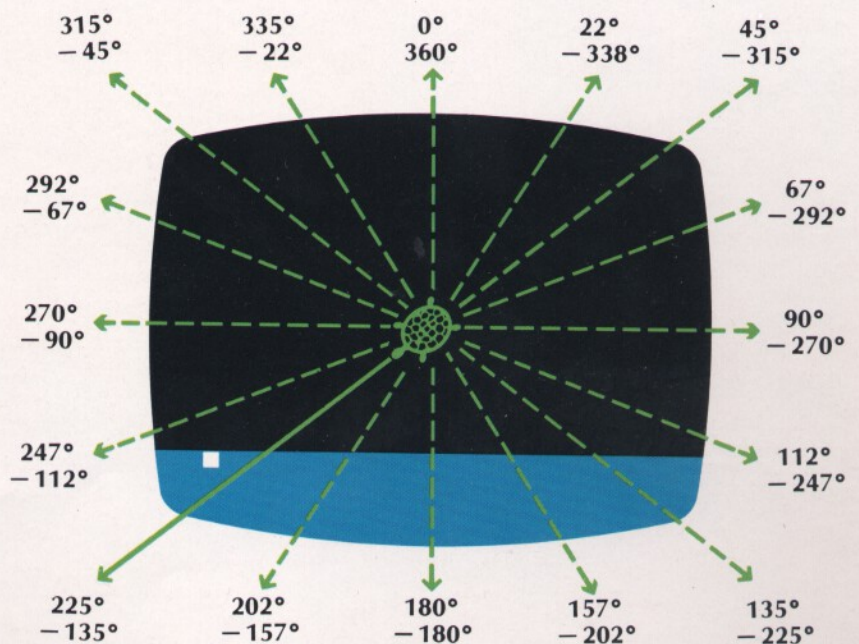


Figure 10-5 Calculating Directional Angles

In turtle geometry, the turtle always moves a given distance relative to its current position and direction. Rather than moving to an (X,Y) location, the turtle can be told to turn and/or move a certain distance.

The subcommands that use turtle geometry are:

DRAW
FILL
GO
TURN
TURNT0

■ Some important things to remember about the two systems are that the *Cartesian coordinates X and Y specify an exact location on the graphics screen*, whereas *turtle geometry specifies direction and distance relative to the turtle*. These two systems are interchangeable; they can be used on the graphics screen simultaneously. The turtle acts like a cursor in both systems, marking the current position on the screen. When graphics mode is initialized, the X and Y coordinate positions and the turtle's directional angle THETA are all set to 0. In other words, when graphics mode is initialized, the turtle is placed in the center of the screen, pointing straight up. ■

GRAPHICS SUBCOMMANDS

THE PEN SUBCOMMAND

All drawing is performed by the graphics *pen*. But this is one pen you will never see; it is invisible!

The graphics pen has four *colors*. One of these colors is an *uncolor*, which is the same as the background color. The other three colors are *red*, *yellow*, and *blue*.

The PEN command assigns a color to your graphics pen. The format for the PEN command is:

GR: PEN *color*

Examples:

GR: PEN RED
GR: PEN YELLOW
GR: PEN BLUE
GR: PEN ERASE

When the graphics screen is first opened, the color is automatically yellow. The color remains set until another PEN graphics command is given or until the graphics screen is closed.

The ERASE color is the background color. GR: PEN ERASE is useful for erasing mistakes. If you draw an incorrect line, change the PEN color to ERASE and redraw the same line.

This is how GR: PEN ERASE works. Type in the following command:

GR: DRAWTO 20,20 and press **RETURN**

The screen should display:



Figure 10-6 GR: DRAWTO 20,20

To erase this line, type:

```
GR: PEN ERASE
GR: GOTO 0,0
GR: DRAWTO 20,20
```

The first line changes the PEN color to the background color. The following two lines (which you won't understand as of yet), draw over the yellow line. And guess what happens when you RUN the program; the line disappears!

There is one more type of PEN subcommand: the PEN UP subcommand. PEN UP tells the computer to lift the imaginary pen off the screen. This is different than PEN ERASE because PEN ERASE erases over other graphics, whereas PEN UP moves over the graphics screen without erasing the display.

■ The PEN subcommand does not draw anything. It simply selects a color or raises or lowers the imaginary pen. Therefore, when you issue a PEN subcommand, you won't see any change in the screen display until you make the pen draw with one of the following subcommands. ■

After using the PEN UP subcommand, you must assign a PEN color with the PEN subcommand before the PEN will show on the graphics screen again.

SUBCOMMANDS USING CARTESIAN COORDINATES

The GOTO Subcommand

The GOTO subcommand moves the turtle to a specified pair of Cartesian coordinates (X,Y) without drawing a line. The turtle draws a dot at that location.

The format for the GOTO subcommand is:

GR: GOTO X,Y

where: X is the X-axis coordinate

Y is the Y-axis coordinate

For example, type in:

GR: GOTO 20,20 and press **RETURN**

If you look closely at your screen, you should see a small dot in the upper right part of the screen.

Following is a program that really demonstrates the GOTO subcommand. *This program plots random units on the screen with the GR: GOTO command. Type:*

```
10 *START
20 C: #X = ?\75
30 C: #Y = ?\45
40 C: #N = #N + 1
50 C(#N = 2): #Y = - (#Y)
60 C(#N = 3): #X = - (#X)
70 C(#N = 3): #Y = - (#Y)
80 C(#N = 4): #X = - (#X)
90 GR: GOTO #X, #Y
100 C(#N = 4): #N = 0
110 J: *START
```

Look at the diagram below. The screen is divided into four quadrants showing when X and Y must be positive or negative.

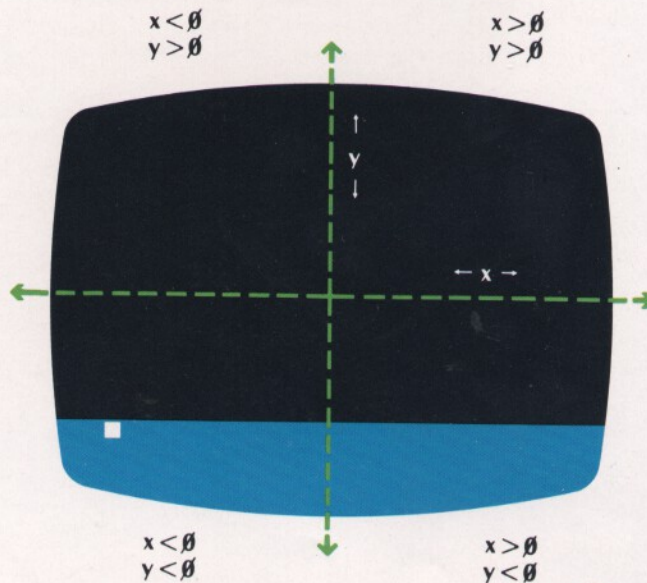


Figure 10-7 The Four Quadrants of the GRAPHICS Screen

In the above program, the random number operator `?`, followed by the modulo operator `\`, forces a positive number. The dots would be distributed where both `#X` and `#Y` are always positive: the upper right quadrant. Lines 40 through 80 adjust the values of `#X` and `#Y` so that some are positive and some are negative, preventing all the dots from appearing in the upper right quadrant.

When you RUN the program, the screen slowly fills up with yellow dots.

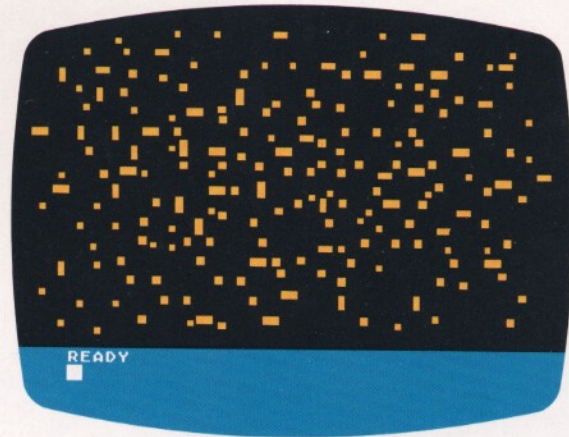


Figure 10-8 Filling the Screen With Dots Using `GOTO`

The `DRAWTO` Subcommand

The `DRAWTO` subcommand moves the turtle from the current cursor position to the specified point while drawing a line. The point is defined by an `(X,Y)` coordinate pair.

The format for the `DRAWTO` subcommand is:

GR: `DRAWTO X,Y`

where: `X` is the X-axis coordinate

`Y` is the Y-axis coordinate

Examples:

GR: `DRAWTO 15,35`

GR: `DRAWTO #J,#K/2`

The line is drawn in the color of the current pen color. If the pen is UP, then no line is drawn.

The following program draws a box using the `PEN` and `DRAWTO` subcommands. Clear memory and type:

10 GR: `PEN RED; DRAWTO 20,0`

20 GR: `PEN BLUE; DRAWTO 20,20`

30 GR: `PEN RED; DRAWTO 0,20`

40 GR: `PEN BLUE; DRAWTO 0,0`

RUN the program. You should see this on the screen:

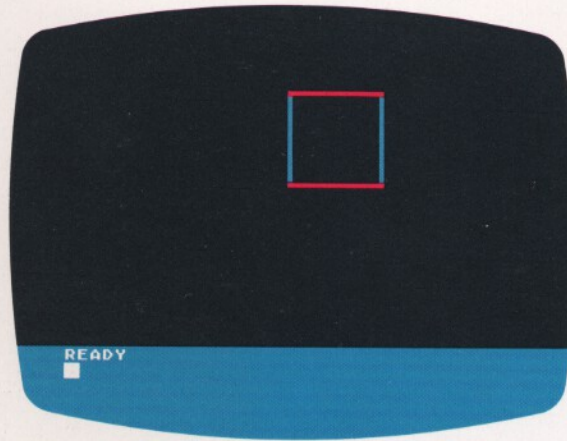


Figure 10-9 Drawing a Box

■ Multiple commands can follow a GR: command if they are separated by a semicolon. ■

The box is drawn counterclockwise, starting at the lower left corner. Notice that DRAWTO begins each line from where the previous line ended.

To begin a line from a location other than the end of the previous line (the current turtle position) use the GOTO subcommand to relocate the turtle. Then draw the line to the desired point:

GR: GOTO A,B

GR: DRAWTO X,Y

where: A,B is the desired starting point

X,Y is the desired ending point

Experiment on your own to see how GOTO can really change things.

The FILLTO Subcommand

The FILLTO subcommand draws a line to the specified point, while filling in the blank regions to the *right* of the line with the current pen color.

The format for the FILLTO subcommand is:

GR: FILLTO X,Y

where: X,Y are the coordinates of the destination point.

Examples:

GR: FILLTO 40,-10

GR: FILLTO 20,#C

Both the line and the fill space are drawn in the current pen color. If the pen is UP, then no line or fill space is drawn.

FILLTO fills blank space to the right of the turtle until a graphic line or figure is reached. At that point, the fill color stops. If there is nothing to block the fill color, FILLTO fills out beyond the right edge of the screen, wrapping around and filling in space from the left side.

To test the FILLTO subcommand, LIST the example "box" program from DRAWTO. Draw a line from the lower left corner of the box (point -20,0) to the upper right corner of the box (20,20) and fill the space to the right with yellow. This can be done with three subcommands. Add the following statements to the end of the program:

```
50 GR: GOTO -20,0
60 GR: PEN YELLOW
70 GR: FILLTO 19,20
```

RUN the program. The screen should display:

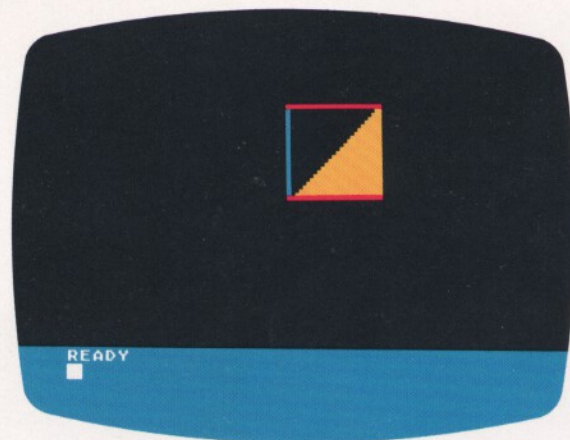


Figure 10-10 Filling With FILLTO

Look closely at the box. The right side of the box is partly covered with the yellow fill.

SUBCOMMANDS USING TURTLE GEOMETRY

The TURNTO Subcommand

The TURNTO subcommand sets the angle THETA to a value in degrees. This, in effect, points the turtle in the specified THETA angle.

The format for the TURNTO subcommand is:

```
GR: TURNTO degree angle
```

Example:

```
GR: TURNTO 90
GR: TURNTO 180
```

If the turtle is positioned at the origin facing 0 degree, the command GR: TURNTO 90 turns the turtle 90 degrees to the right. THETA now equals 90:

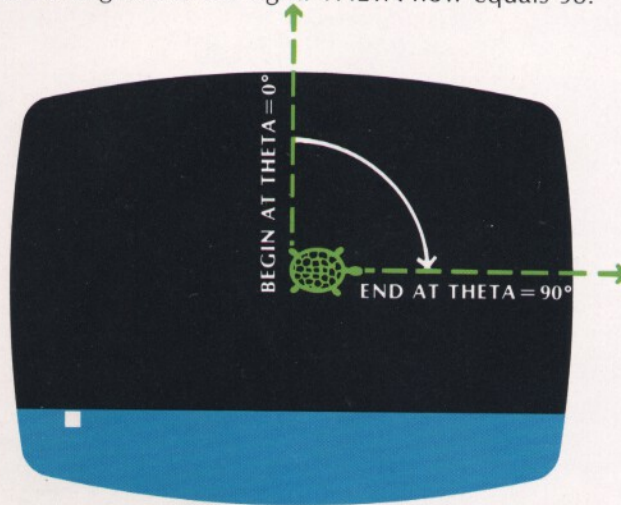


Figure 10-11 Turning the Turtle Toward 90 Degrees

With the turtle facing 0 degree, the command GR: TURNTO -45 turns the turtle 45 degrees to the left. THETA now equals -45:

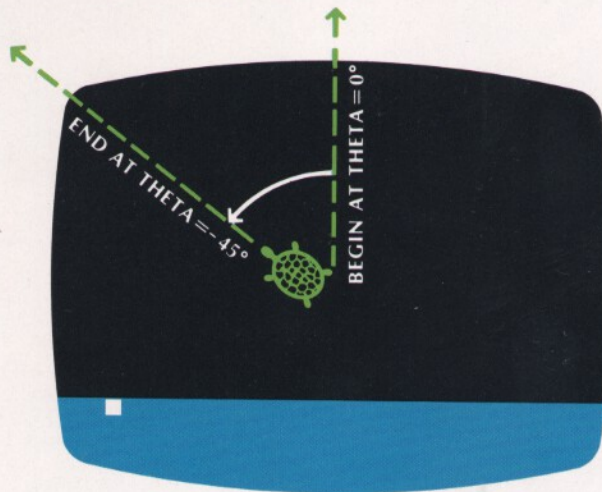


Figure 10-12 Turning the Turtle Toward -45 Degrees

The TURNTO subcommand, like the PEN subcommand, shows no visible effect until a line is drawn with another subcommand.

The TURN Subcommand

This subcommand increments the angle THETA by the number of specified degrees (THETA = THETA + increment).

The format of the TURN subcommand is:

GR: TURN *increment*

Examples:

GR: TURN 45

GR: TURN -60

GR: TURN 45 turns the turtle 45 degrees to the right from whatever direction it is currently facing. If the turtle is positioned facing 60 degrees, GR: TURN 45 would position the turtle facing 105 degrees ($\text{THETA} = 60 + 45$).

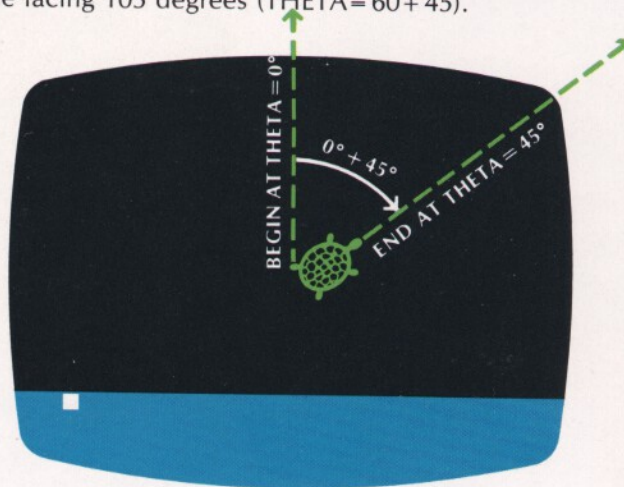


Figure 10-13 Turning the Turtle 45 Degrees to the Right

GR: TURN -60 turns the turtle 60 degrees to the left of its current direction.

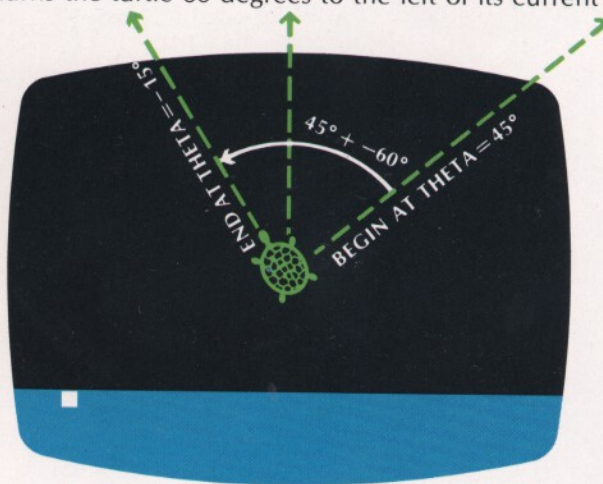


Figure 10-14 Turning the Turtle 60 Degrees to the Left

Here is a short example:

```
10 GR: GOTO 0,0
20 GR: TURNT0 0
30 GR: DRAW 30
40 GR: TURN 45
50 GR: DRAW 40
```

Lines 10 through 30 draw a line from the origin straight up to point 0,40. Line 40 adds 45 degrees to THETA (THETA = 0 + 45). Line 50 draws a line 40 units long from point (0,40) in the direction of THETA.

Your screen should look like this:

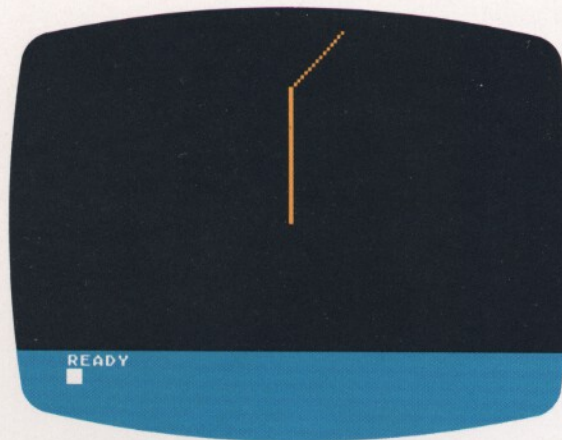


Figure 10-15 TURN Subcommand: Program Run

The DRAW Subcommand

The DRAW subcommand draws a line the number of units specified in the direction of THETA in the current pen color.

The format for the DRAW subcommand is:

GR: DRAW *line length*

Examples:

GR: DRAW 20

GR: DRAW #X

The line is drawn in the current PEN color.

The following program demonstrates the GOTO, TURNT0, and DRAW subcommands. Type:

```
10 *START
20 GR: GOTO 0,0
30 GR: TURNT0 #X
40 GR: DRAW 30
50 C: #X = #X + 10
60 J: *START
```

The program is one continuous loop. Line 20 repositions the turtle at the origin in each loop. Line 30 faces the turtle #X degrees. Line 40 draws a line from the origin facing #X degrees and 30 units long. Line 50 increments #X by 10 in each loop.

RUN the program. This is what you should see:



Figure 10-16 DRAW Subcommand: Program Run

DRAW lets you draw a line in a *direction* without having to specify exact Cartesian coordinates.

The GO Subcommand

The GO subcommand moves the turtle forward the number of units specified in the current THETA angle and plots a point.

The format for the GO subcommand is:

GR: GO *distance between points*

Examples:

GR: GO 10

GR: GO #L

The point is drawn in the current pen color.

We can demonstrate the GO subcommand by changing a couple of lines from the previous example program. Replace lines 40 and 50 with:

```
40 GR: GO 30
```

```
50 C: #X=#X+1
```

LIST and RUN the program. The program draws a point 30 units from the origin every degree for 360 degrees (a circle). Your screen should look like this:

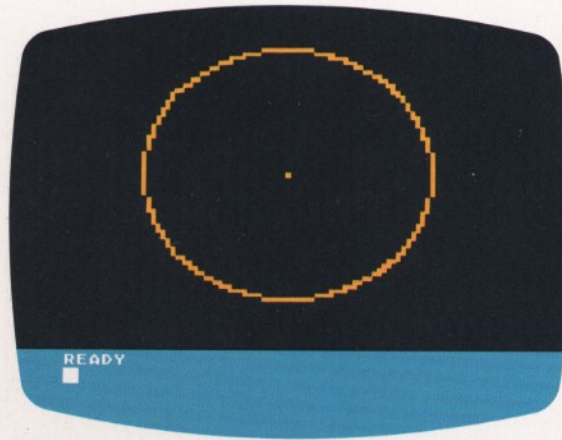


Figure 10-17 GO Subcommand: Program Run

The FILL Subcommand

The FILL subcommand draws a line while it moves the turtle the number of units specified in the current THETA angle. As FILL draws a line it also fills any blank regions to the right of the line with the current pen color.

The format for the FILL subcommand is:

GR: FILL *line length*

Examples:

GR: FILL 30

GR: FILL #F

If the pen is UP, then no line or fill color is drawn. Following is an example of the FILL subcommand: Enter:

```
GR: GOTO 0,0
GR: TURNT0 0
GR: DRAW 40
GR: TURN 90
GR: DRAW 40
GR: TURN 90
GR: DRAW 40
GR: TURN 90
GR: DRAW 40
GR: TURN 90
```

(This draws a square.)

GR: FILL 40

(This fills the square with yellow.)



Figure 10-18 Using the FILL Subcommand

■ In each of the turtle geometry subcommands, if the line length specified is negative, the turtle will move backward instead of forwards. ■

REPEATING GRAPHICS COMMANDS

You can repeat one or more GRAPHICS subcommands by enclosing them within parentheses, (). A numeric expression placed in front of the left parenthesis tells the computer how many times to repeat the GRAPHICS subcommands.

The format for repeating GRAPHICS subcommands is:

GR: *numeric expression*(*subcommands*)

where: *numeric expression* represents the number of times to repeat the subcommands

subcommands are the subcommands to be repeated. If there are more than one, they must be separated by semicolons

Examples:

GR: 180(DRAW 1; TURN 2)

GR: 10(DRAW 20; TURN - 36)

GR: 360(DRAW 1; TURN 1)

Type in:

GR: 4(DRAW 20; TURN 90)

This command repeats itself four times to draw a box. RUN the program and watch.



Figure 10-19 Drawing a Box With Repeating GRAPHICS Commands

This next example draws a star by repeating the commands five times:

```
10 R: STAR  
20 GR: 5(DRAW 25; TURN 144)  
30 E:
```

A RUN of this simple program produces:



Figure 10-20 Drawing a Star With Repeating GRAPHICS Commands

HOME, SWEET HOME PROGRAM

The following program draws a simple picture:

```
10 R: PICTURE
20 GR: CLEAR
30 GR: GOTO - 25, - 10
40 GR: PEN BLUE
50 U: *BOX
60 GR: TURNT0 30; GO 25
70 GR: PEN RED
80 U: *TRIANGLE
90 GR: GOTO 0,25
100 GR: PEN YELLOW
110 U: *MOON
120 GR: GOTO - 40,25
130 U: *STAR
140 GR: GOTO - 10,35
150 U: *STAR
160 GR: GOTO 35,20
170 U: *STAR
299 E:
400 *BOX
410 GR: TURNT0 90
420 GR: 3(DRAW 25; TURN 90)
430 GR: FILL 25
440 E:
500 *TRIANGLE
510 GR: TURNT0 150
520 GR: 2(DRAW 25; TURN 120); FILL 24
530 E:
600 *MOON
610 GR: TURNT0 70
620 GR: 38 (DRAW 1; TURN 5)
630 GR: TURNT0 50
640 GR: 5(TURN - 20; FILL 5)
650 E:
700 *STAR
710 GR: 5(DRAW 10; TURN 144)
720 E:
```

RUN the program. Does your picture look like this?:



Figure 10-21 Home, Sweet Home Program Run

SECTION 10 SUMMARY

In graphics mode, PILOT can draw lines and figures in red, yellow, or blue. *Graphics mode* is characterized by the *graphics screen*, which is all black except for four lines at the bottom of the screen. This blue strip is called the *text window*. It displays up to four lines of text at a time.

The command that controls all of the PILOT graphics capabilities is the GRAPHICS command, in the format:

GR: *subcommand*

Subcommands are commands that *help* the GRAPHICS command tell the computer what to do while in the graphics mode.

To open the graphics screen, type: **GR:** followed by *any subcommand except QUIT*. To close the graphics screen and return to text mode, type: **GR: QUIT** and press **RETURN**.

The other subcommands available are:

CLEAR
DRAW
DRAWTO
FILL
FILLTO
GO
GOTO
PEN
QUIT
TURN
TURNTO

GR: CLEAR *clears* the graphics screen and the text window. **GR: CLEAR** is a good command to use to open the graphics screen.

The subcommands may be divided into two categories: those that use Cartesian coordinates and those that use turtle geometry.

Subcommands using *Cartesian coordinates* specify exact point locations on the screen with the (X,Y) coordinates: X represents the distance from the point to the origin (screen center) along the X (horizontal) axis; Y represents the distance from the point to the origin along the Y (vertical) axis.

The subcommands that use Cartesian coordinates are:

DRAWTO
FILLTO
GOTO

Subcommands using *turtle geometry* specify directions that the turtle will turn, and the distance it moves in that direction. The direction is a polar angle, called THETA. THETA can be a degree from 0 to 360. Figure 10-5 shows some of the angles of THETA.

Subcommands that use turtle geometry are:

DRAW
FILL
GO
TURN
TURNTO

There are three subcommands that don't use Cartesian or turtle geometry: CLEAR, PEN, and QUIT. GR: CLEAR clears the graphics screen, and GR: QUIT returns the computer to text mode.

ATARI PILOT has an imaginary pen that draws on the screen. This pen has three colors: red, yellow, and blue. GRAPHICS automatically uses yellow unless you change the pen color with the PEN command. The following is an example changing the pen color to red.

Example:

GR: PEN RED

The PEN can also erase colors and lines; simply choose the ERASE color and redraw over the area you want erased.

GR: PEN ERASE

If you want to move the pen across the screen without drawing any lines, specify UP:

GR: PEN UP

This is different from ERASE because it doesn't erase anything.

The GOTO subcommand moves the turtle to a specified Cartesian coordinate (X,Y) without drawing a line. The format for the GOTO subcommand is:

GR: GOTO X,Y

Examples:

GR: GOTO 5,10
GR: GOTO -40,40

The DRAWTO subcommand draws a line to a specified Cartesian coordinate (X,Y). The format for DRAWTO is:

GR: DRAWTO X,Y

Examples:

GR: DRAWTO 15, 65
GR: DRAWTO #A, #B

The FILLTO subcommand draws a line to a specified Cartesian coordinate (X,Y) while filling the blank regions to the right of the line with the current pen color. The format for FILLTO is:

GR: FILLTO X,Y

Examples:

GR: FILLTO - 10, - 20
GR: FILLTO 5, L

The TURNTO subcommand sets THETA to a value in degrees and turns the turtle to the specified degree angle. The format for TURNTO is:

GR: TURNTO *degrees*

Examples:

GR: TURNTO 90
GR: TURNTO - 60

The TURN subcommand increments THETA by the number of degrees specified. The format for TURN is:

GR: TURN *increment*

Examples:

If THETA = 45 then
GR: TURN 45 → THETA = 90
GR: TURN - 60 → THETA = - 15

The DRAW subcommand draws a line for the length specified, in the direction of THETA. The format for DRAW is:

GR: DRAW *line length*

Examples:

GR: DRAW 20
GR: DRAW #X

The GO subcommand moves the turtle a specified length in the direction of THETA. The format for GO is:

GR: GO *number of units from starting point*

Examples:

GR: GO 20
GR: GO #L
GR: GO - 5

The FILL subcommand draws a line the length specified, in the direction of THETA, while filling in the blank regions to the right with the current pen color. The format for FILL is:

GR: FILL *line length*

Examples:

GR: FILL 30
GR: FILL # A
GR: FILL - 15

In all the turtle geometry subcommands, if the line length is negative, the turtle moves backward instead of forward.

SECTION 10
QUIZ

1. The graphics screen is all black except for a strip of blue at the bottom of the screen, called the _____.
2. The GRAPHICS command, followed by any subcommand except QUIT, puts the computer into graphics mode. TRUE or FALSE?
3. To close the graphics screen, type: _____.
4. Put the subcommands on the left under the coordinate system they respond to:

Subcommands	Cartesian Coordinates	Turtle Geometry
--------------------	------------------------------	------------------------

DRAW

DRAWTO

FILL

FILLTO

GO

GOTO

TURN

TURNTO

5. The _____ subcommand assigns the color of the graphics pen.

6. Match the subcommands with their functions:

GOTO

Turn turtle to THETA angle

DRAWTO

Move turtle specified length in THETA direction

FILLTO

Draw line of specified length in THETA direction and fill space to right

TURNTO

Move turtle to point X,Y

TURN

Increment THETA angle by specified degrees

GO

Draw line of specified length in THETA direction

DRAW

Draw line to point X,Y and fill space to right

FILL

Draw line to point X,Y

ANSWERS

- 1. Text window
- 2. TRUE
- 3. GR: QUIT

4. **Cartesian Coordinates**

DRAWTO
GOTO
FILLTO

Turtle Geometry

DRAW
GO
FILL
TURN
TURNT0

- 5. PEN

- 6. GOTO

DRAWTO

FILLTO

TURNT0

TURN

GO

DRAW

FILL

Turn turtle to THETA angle

Move turtle specified length in THETA direction

Draw line of specified length in THETA direction and fill space to right

Move turtle to point X,Y

Increment THETA angle by specified degrees

Draw line of specified length in THETA direction

Draw line to point X,Y and fill space to right

Draw line to point X,Y

ADVANCED PROGRAMMING WITH GRAPHICS

GRAPHICS VARIABLES

PILOT has four variables that display information about the graphics turtle. These variables can not be assigned or changed by the user. They can be displayed or used in computations of numeric expressions.

%X

The variable %X returns the current value of the X coordinate of the graphics turtle.

Example:

```
T: %X  
25
```

%Y

The variable %Y returns the current value of the Y coordinate of the graphics turtle.

Example:

```
T: %Y  
20
```

%A

The variable %A returns the current value of THETA.

Example:

```
T: %A  
90
```

%Z

The variable %Z returns the current number equivalent of the screen color at the current turtle location.

```
0 = ERASE (background)  
1 = RED  
2 = YELLOW  
3 = BLUE
```

Example:

```
GR: GO 0  
T: %Z  
2
```

■ Always precede the T: %Z command with a GR: GO 0 command. ■

Enter the following program:

```
10 GR: PEN RED
20 GR: TURN 50
30 GR: DRAW 40
```

To find the values of %X, %Y, %A, and %Z, enter a TYPE command for each variable:

```
T: %X
31
```

```
T: %Y
26
```

```
GR: GO 0
T: %Z
1
```

The above values tell us that the turtle is located at the (31,26) coordinate. The THETA angle is 50 degrees. The turtle at the point is RED.

MINI-QUIZ

1. The variables %X, %Y, %A, %Z can be changed at any time by the user. TRUE or FALSE?
2. To display one of the above variables, you must use the _____
_____ command.

ANSWERS

1. FALSE
2. TYPE, or T:



APPENDIX A

CONNECTING YOUR ATARI COMPUTER SYSTEM

CONCEPTS INTRODUCED

Connecting Your ATARI Computer System
Inserting PILOT

Turning On PILOT in Your ATARI Computer Without a Disk Drive

Turning On PILOT in Your ATARI Computer With an ATARI 810 Disk Drive

Turning Off Your ATARI Computer System

CONNECTING YOUR ATARI COMPUTER SYSTEM

You may connect other machines to your computer besides the television set. These machines are called *devices*. Some of the devices that can be connected to your computer are the **ATARI 810™ Disk Drive** (Figure A-1), the **ATARI 410™ Program Recorder** (Figure A-2), and the **ATARI 820™, 822™, and 825™ printers** (Figure A-3). When attached to the computer, these devices assist in remembering programs.



Figure A-1 The ATARI 810 Disk Drive

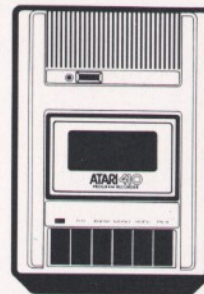


Figure A-2 The ATARI 410 Program Recorder

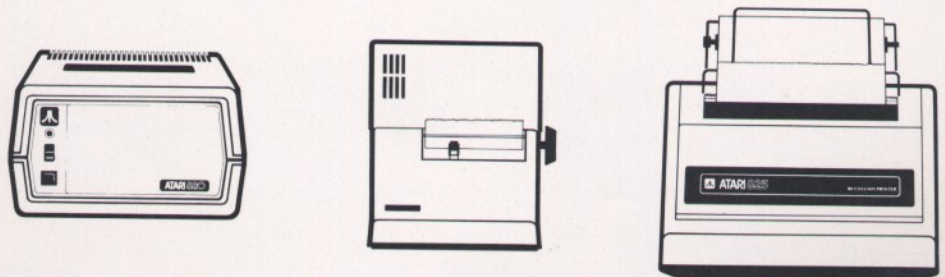


Figure A-3 The ATARI 820, 822, and 825 Printers

The first step in getting started is to learn how to assemble your computer system. This is easy if you read this section carefully.

To hook up the computer with one or more devices, start by reading the introduction, unpacking, system requirements, set-up and start-up instructions of the manuals that apply to your system. **Skip the sections referring to the BASIC language. BASIC language commands will not work when the PILOT cartridge is inserted.** Read:

- ATARI 400™ Operator's Manual or ATARI 800™ Operator's Manual
- ATARI 810™ Disk Drive Operator's Manual
- ATARI 410™ Program Recorder Operator's Manual
- ATARI 820™ Printer Operator's Manual
- ATARI 822™ Thermal Printer Operator's Manual
- ATARI 825™ 80-Column Printer Operator's Manual

Following are several diagrams showing you different ways to connect your system:

System with ATARI 400/800 and ATARI 410 Program Recorder

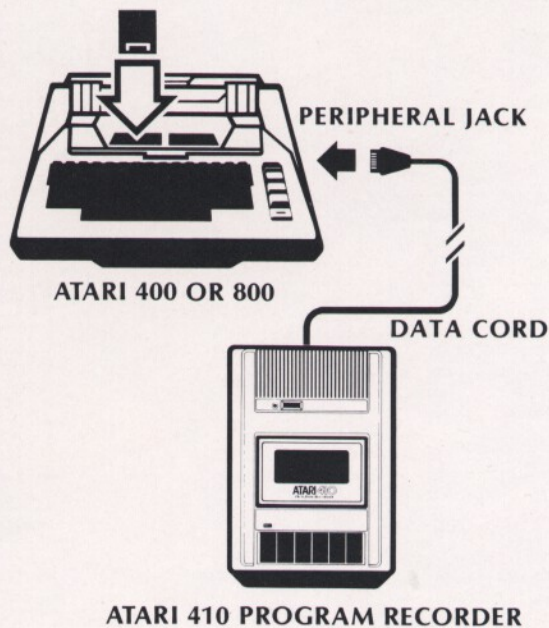


Figure A-4 ATARI 400/800 Computer With Program Recorder

System with ATARI 800 and ATARI 820 40-Column Printer, or ATARI 810 Disk Drive

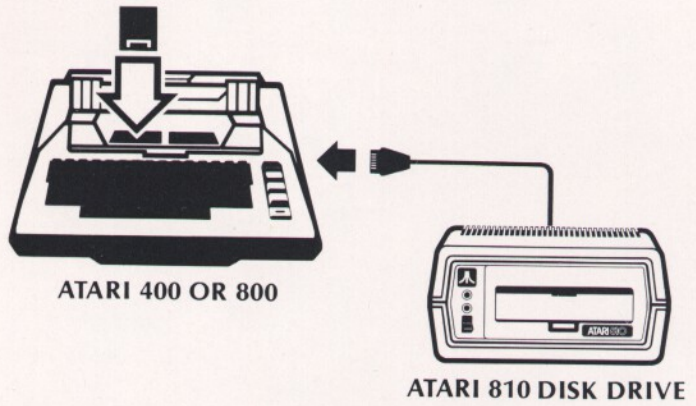


Figure A-5 ATARI 400/800 Computer With Disk Drive

System with ATARI 400/800, ATARI 820 Printer, and ATARI 410 Program Recorder

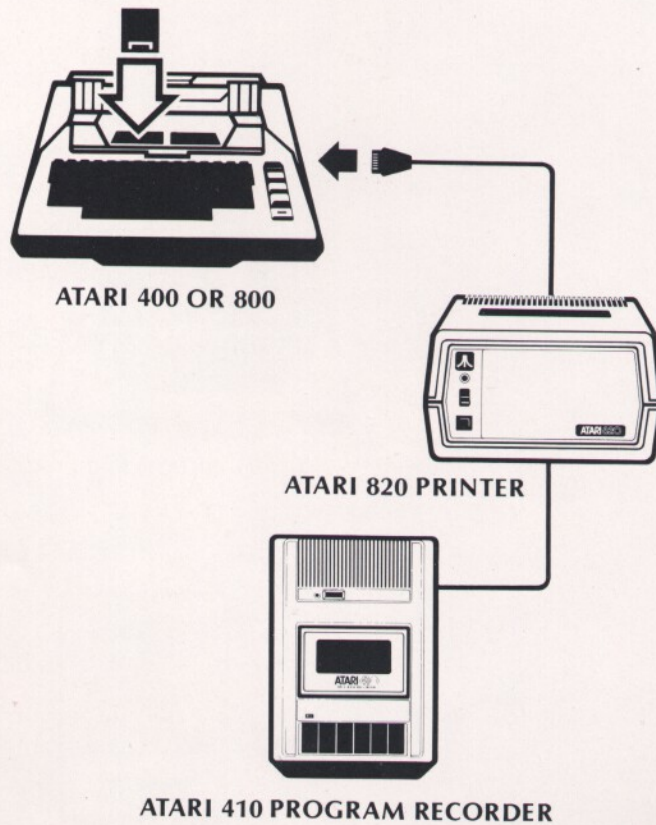


Figure A-6 ATARI 400/800 Computer With Program Recorder and Printer

System with ATARI 800, ATARI 810 Disk Drive, and ATARI 820 40-Column Printer

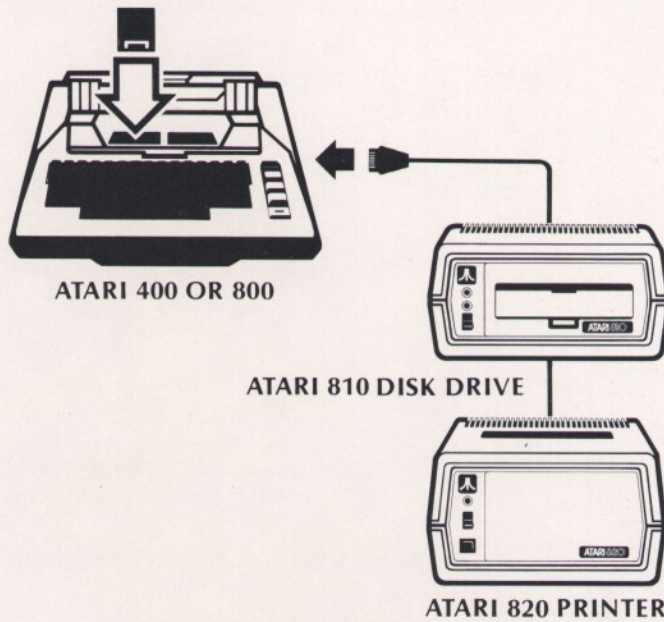


Figure A-7 ATARI 400/800 Computer With Disk Drive and Printer

System with ATARI 800, ATARI 810 Disk Drive, ATARI 820 Printer, and ATARI 410 Program Recorder.

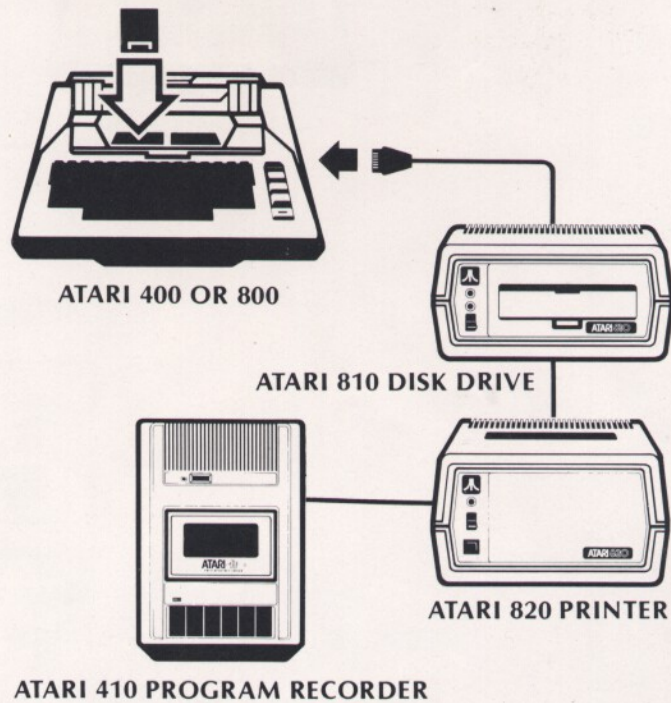


Figure A-8 ATARI 400/800 Computer With Disk Drive, Program Recorder, and Printer

INSERTING PILOT

To insert the ATARI PILOT cartridge into your **ATARI 400™/800™ Personal Computer System**:

1. Pull the lever on the top of your computer marked PULL OPEN. The top of the computer will spring open, and if the television is turned on, the screen will fill with static. Don't panic; screen static is normal whenever the top of the computer is open.
2. Hold the ATARI PILOT cartridge with the label toward you and the opening at the bottom. Insert the cartridge into the proper slot of either the ATARI 400 or ATARI 800 Personal Computer. Make sure the cartridge is all the way in and secure. (Use the left cartridge slot on the ATARI 800.)
3. Close the computer top all the way. When you turn on the computer, the computer won't operate if the top is even slightly open. The television screen will display this message:

ATARI PILOT (C) COPYRIGHT ATARI 1980

READY



TURNING ON PILOT IN YOUR ATARI COMPUTER WITHOUT A DISK DRIVE

Follow the steps below to turn your computer on correctly.

1. Make sure that your television switch box is connected properly. It should be switched to the GAME or COMPUTER position. Turn on and tune your television set to channel 2 or 3 and set your channel switch to the same number. Be sure the PILOT cartridge is inserted.
2. Push the power switch on the right side of the computer forward to the ON position.
3. The television screen should display:

ATARI PILOT (C) COPYRIGHT ATARI 1980

READY



The system is now turned on.

■ If the screen is full of static instead of the blue screen display, check all connections and make sure the top lid of the computer is closed securely. ■

TURNING ON PILOT IN YOUR ATARI COMPUTER WITH AN ATARI 810 DISK DRIVE

1. Make sure that the television switch box is connected properly. Make sure that both the television and computer are tuned to channel 2 or 3 and that the PILOT cartridge is inserted.
2. Make sure that the cords from the computer and the ATARI 810 Disk Drive are properly and securely connected.
3. The computer should be turned OFF and there should **NOT** be a diskette in the disk drive.

-
4. Turn the disk drive power switch on the front of the disk drive to ON. Both red lights on the front should turn on. The sounds you now hear are normal and will stop shortly. Wait until the BUSY light (top light) goes off before proceeding to step 5. *NEVER interrupt the disk drive when the BUSY light is on.*
 5. Hold the diskette with the label facing **up** and the notch on the **left** side. **DO NOT TOUCH THE SHINY SURFACE OF THE DISKETTE** that shows through the opening in the protective jacket. If you touch the shiny surface, you may damage the diskette.
 6. Push in the door release lever located beneath the disk drive door. Insert the diskette, held as described above, until you hear it click securely into place.
 7. Close the disk drive door.
 8. Turn on the power switch on the right side of the computer. The disk drive should start making noises, which indicates that the computer is reading in parts of the Disk Operating System (DOS). The DOS is required in order for the computer to store and retrieve programs on the diskette.
 9. This message will be displayed:

ATARI PILOT (C) COPYRIGHT ATARI 1980

READY



The system is now turned on.

If you encounter any problems turning on your system with the disk drive, read the section called Don't Panic! in Appendix B, Speaking To Devices.

TURNING OFF YOUR ATARI COMPUTER SYSTEM

To turn off your computer system, turn the switch on the right side of your computer to OFF. Turn your television set to OFF.

If you have an ATARI 810 Disk Drive, you should first:

1. Open the disk drive door by pressing the door release lever beneath the door.
2. Remove the diskette by pulling it straight out. Return the diskette to its envelope.
3. Turn the disk drive power switch to the OFF position.

Your system is now turned off.

APPENDIX B

SPEAKING TO DEVICES

CONCEPTS INTRODUCED

- The ATARI 810 Disk Drive
 - Speaking to Your ATARI 810 Disk Drive
 - Saving a Program on a Diskette
 - Write-Protecting a Diskette
 - Loading a Program From a Diskette
 - Merging Programs
 - The DOS Utility
 - The Disk Directory
 - Using the DOS Disk Directory
 - Creating Your Own Disk Directory
 - Returning to PILOT From the DOS Utility
 - Don't Panic! (How To Handle Trouble With the Disk Drive)
 - Screen Displays "Boot Error"
 - Busy Light on Disk Drive Does Not Go Off
 - The Computer Does Not Respond
- The ATARI 410 Program Recorder
 - Speaking to Your Program Recorder
 - Saving a Program on a Cassette Tape
 - Write-Protecting a Cassette Tape
 - Loading a Program From a Cassette Tape
 - Merging Programs
 - A Method to Manage Programs on Cassette Tapes
 - Saving the Cassette Directory
 - Adding Programs to the Directory and the Tape
- The ATARI Printers
 - Printing a Program

The ATARI 400/800 Personal Computer System is made up of several *devices* that transfer information from one to another. They include:

- The keyboard
- The disk drive
- The program recorder
- The printer

Three of these devices handle information in program-size chunks. They are: the *disk drive*, the *program recorder*, and the *printer*.

Because the computer is just a machine that is run by electricity, when it is turned off the electricity stops and it "forgets" everything. This can be a problem if you want the computer to remember something after it is turned off. To solve this problem, a device such as a program recorder, disk drive, or printer may be attached to the computer to become part of the system. The program recorder, when instructed to do so, records the information in the computer's memory onto a cassette tape (Figure B-1), just like music is recorded onto a cassette tape. Or, if you have a disk drive as part of your system, the disk drive, when instructed to do so, records information from the computer's memory onto a diskette (Figure B-2). A diskette is a flexible 5¼ inch magnetic diskette, much like a miniature record.

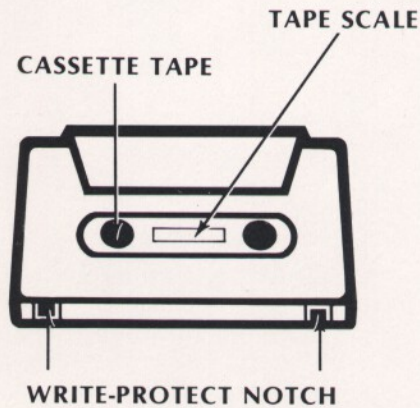


Figure B-1 Cassette Tape

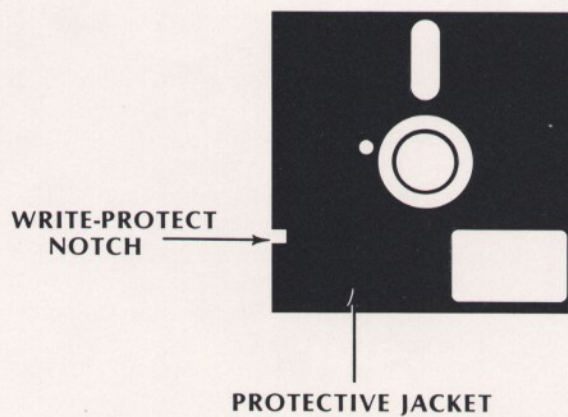


Figure B-2 Diskette

When information from the computer's memory is recorded onto a cassette tape or diskette, the computer can be turned off without losing the recorded information. Later, when the computer is turned on, you can tell the computer to get the information from the cassette tape or diskette and put it back in its memory.

Another alternative is to send information from memory to the printer. The printer is a device that will type on paper whatever you tell it to print.

THE ATARI 810 DISK DRIVE

The disk drive unit stores programs and data on diskettes. Each diskette can store many, many programs on it. A total of four ATARI 810 disk drives can be attached to the ATARI computer at one time, allowing you to store and retrieve a multitude of programs.

SPEAKING TO YOUR ATARI 810 DISK DRIVE

The program (also called a *file*) currently in the computer's memory can be saved on a diskette and *loaded* or retrieved at a later time.

Saving a Program on a Diskette

1. The disk drive and computer should be turned on following the steps listed in Appendix A, Turning On Your ATARI Computer With a Disk Drive.
2. Make sure a diskette is inserted into the disk drive.
3. Enter a program into the computer.
4. Think of a name for your program. A program name, or file name, must begin with a letter, optionally followed by either letters, numbers, or a combination of both. The computer remembers only the first eight characters of the program name unless it is followed by a period; it will also remember up to three characters following the period. *Embedded spaces are not allowed. Graphic characters are ignored. Anything following a punctuation mark other than a period is ignored.*

Here are some examples of correct and incorrect program names:

Correct

PROGNAME
CAP.81
ATARI

Incorrect

PROG NAME
81.CAP
ATARICOMPUTERS

If you choose an incorrect program name, the computer will respond with a *** I/O ERROR 165 *** error message when the command described below is given.

5. The SAVE command saves the program currently in memory on a diskette. Type:

SAVE Dn: *Program name* **RETURN**

where: **D** is the disk drive

n specifies which disk drive (1-4). Because up to four disk drives may be attached, the computer must know now which one to use. If **n** is not specified, it defaults to 1.

Program name is the complete program name.

Here are some examples:

SAVE D: CHECKSUM

Save CHECKSUM on Drive#1

SAVE D1: LETTER.1

Save LETTER.1 on Drive#1

SAVE D4: K

Save K on Drive#4

If only part of a program is to be saved, the program lines to be saved must be specified following the program name:

SAVE Dn: *Program name x,y*

where: *x* is the first program line to be saved

y is the last program line to be saved

This allows you to save sections and modules out of an entire program.

Examples:

SAVE D2: LUCKY 13,99 Saves lines 13-99 of LUCKY on Drive#2
SAVE D: MDYF 100,200 Saves lines 100-200 of MDYF on Drive#1

Be careful to include a space between the program name and the first line number.

6. When you press the **RETURN** key the BUSY light will turn on and the disk drive will start making noises as it saves the program. When the program has been saved, the READY message and the cursor will appear on the screen.

Following is an exercise to take you through the steps of saving a program:

1. Turn on the computer and the disk drive if you have not done so already (make sure a diskette has been inserted).

2. Type in the program below:

```
10 R: TEST PROGRAM
20 T:
30 T: THIS PROGRAM TESTS \
40 T: YOUR ABILITY TO
50 T: SAVE AND LOAD A PROGRAM \
60 T: ON A DISKETTE
70 E:
```

3. Type: **SAVE D: TEST** **RETURN**.
4. When the READY message and cursor are displayed on the screen, you have successfully saved it on the diskette. If an error message appears, refer to Appendix E, Error Codes, to determine the cause of the error. Then repeat step 3.

Write-Protecting a Diskette

Write-protecting a diskette prevents any programs or data from being saved on the diskette. To write-protect a diskette, place an adhesive tab or a piece of cellophane tape over the write-protect notch (shown in Figure A-2 in Appendix A.) If you try to save a program on a diskette that is write-protected, the computer responds with an *** I/O ERROR 144 *** message.

If at anytime you wish to "unwrite-protect" a diskette, simply remove the tape from the write-protect notch.

Loading a Program From a Diskette

1. The disk drive and the computer should be turned on following the steps in Appendix A. Turn on your system with the diskette containing the desired program.
2. Type: **NEW** and press **RETURN** to clear memory.
3. Type: **LOAD Dn: program name**
where: **D** specifies the disk drive
n specifies which disk drive (1-4). If *n* is not specified, *n* defaults to 1.
program name is the saved program's name.

Here are some examples:

LOAD D: CHECKSUM

Loads CHECKSUM from Drive#1

LOAD D: LETTER.1

Loads LETTER from disk Drive#1

LOAD D4: K

Loads K from disk Drive#4

If the program is not on the diskette, the computer responds with a *** I/O ERROR 170 *** error message.

Unlike saving programs, partial programs cannot be loaded. *The entire program must be loaded.*

4. LIST, and/or RUN the program.

Following is an exercise to take you through the steps of loading a program.

1. Turn on the computer and the disk drive if you have not done so.
2. Insert the diskette containing the TEST program saved previously.
3. Type: **LOAD D: TEST** **RETURN**.
4. When the READY message appears on the screen your program is loaded into memory.
5. LIST and RUN the program.

Merging Programs

Programs can be merged when loading off a diskette and into memory. *Merge* means to blend together or combine. If you have a program in memory, and you load another program from a diskette into memory without clearing memory first, the two programs will merge. If both programs use the same line numbers, the statements loaded off the diskette have priority and are remembered.

Merging programs can be advantageous or disastrous. Used correctly, you can expand a current program by merging another program into it.

Thus, if the following program is in memory,

```
10 R: NAME PROGRAM
20 T:
30 T: WHAT IS YOUR NAME? \
40 A: $NAME
50 T: HELLO, $NAME! HOW ARE YOU TODAY? \
60 A:
70 M: GOOD,FINE,GREAT,HEALTHY,EXCELLENT
80 TY: GREAT!
90 TN: THAT'S TOO BAD:
100 E:
```

and you load this program from a diskette,

```
100 R: ADDRESS PROGRAM
110 T: WHAT IS YOUR STREET ADDRESS? \
120 A: $STREET
130 T: IN WHAT CITY AND STATE DO YOU RESIDE? \
140 A: $CITYSTATE
150 T: WHAT IS YOUR ZIP CODE? \
160 A: $ZIP
170 T: THANK YOU FOR THE INFORMATION.
180 E:
```

the resulting *merged* program looks like this:

```
10 R: NAME PROGRAM
20 T:
30 T: WHAT IS YOR NAME? \
40 A: $NAME
50 T: HELLO, $NAME. HOW ARE YOU TODAY? \
60 A:
70 M: GOOD,FINE,GREAT,HEALTHY,EXCELLENT
80 TY : GREAT!
90 TN: THAT'S TOO BAD!
100 R: ADDRESS PROGRAM
110 T: WHAT IS YOUR STREET ADDRESS? \
120 A: $STREET
130 T: IN WHAT CITY AND STATE DO YOU RESIDE? \
140 A: $CITYSTATE
150 T: WHAT IS YOUR ZIP CODE? \
160 A: $ZIP
170 T: THANK YOU FOR THE INFORMATION.
180 E:
```

Notice that line 100 in the first program is replaced with line 100 from the second program.

The two programs have become one in memory. If desired, another program may be merged with this one. Merging can be disastrous if the two programs merged cannot function correctly. For instance, if the program in memory looks like this,

```
1 R: ADDITION PROGRAM
5 T: ENTER A NUMBER: \
25 A: #A
50 T: ENTER ANOTHER NUMBER: \
75 A: #B
100 C: #C=#A+#B
125 T: #A+#B=#C
150 E:
```

and you load the TEST program from the diskette,

```
10 R: TEST PROGRAM
20 T:
30 T: THIS PROGRAM TESTS \
40 T: YOUR ABILITY TO
50 T: SAVE AND LOAD A PROGRAM \
60 T: ON A DISKETTE.
70 E:
```

the resulting *merged* program looks like this:

```
1 T: ADDITION PROGRAM
5 T: ENTER A NUMBER: \
10 R: TEST PROGRAM
```

```
20 T:
25 A: #A
30 T: THIS PROGRAM TESTS \
40 T: YOUR ABILITY TO
50 T: SAVE AND LOAD A PROGRAM \
60 T: ON A DISKETTE
70 E:
75 A: #B
100 C: #C=#A+#B
125 T: #A+#B=#C
150 E:
```

This program will not work, as you can plainly see.

■ To avoid merging two programs by mistake always type **NEW** to clear memory before loading a new program. ■

The DOS Utility

Every diskette you use in your ATARI 810 Disk Drive is supplied with a DOS, which stands for **Disk Operating System**. DOS allows you to manage and manipulate program files.

To leave the PILOT language and enter the DOS utility, enter the DOS command in the immediate mode:

Type: **DOS** **RETURN**

When in DOS, the screen displays:

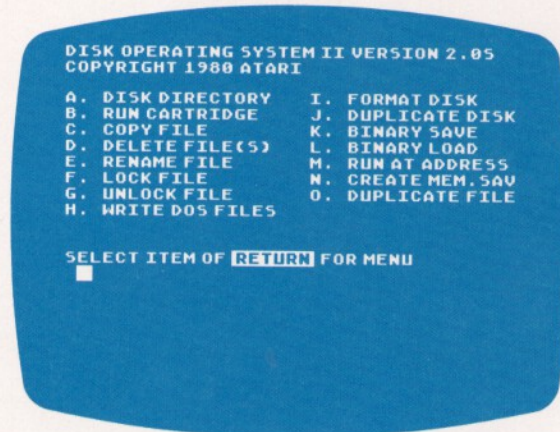
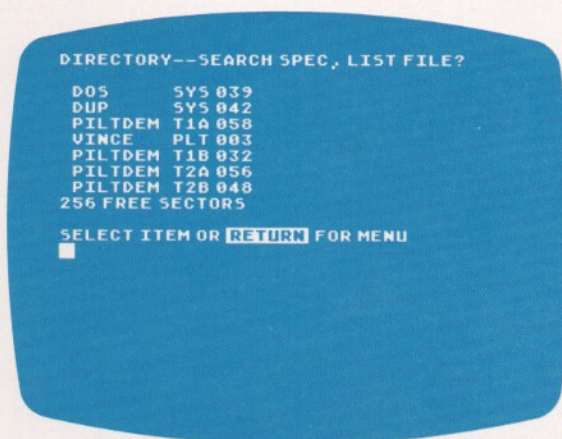


Figure B-3 DOS Utility Program

We will discuss how to use the disk directory and return to PILOT under the DOS utility.

The Disk Directory

Using the DOS Disk Directory. The disk directory is similar to a table of contents. It tells you what files are on the disk and the length of each program file. Below is a sample directory:



```
DIRECTORY--SEARCH SPEC, LIST FILE?  
DOS      SYS 039  
DUP      SYS 042  
PILTDEM  T1A 058  
VINCE    PLT 003  
PILTDEM  T1B 032  
PILTDEM  T2A 056  
PILTDEM  T2B 048  
256 FREE SECTORS  
  
SELECT ITEM OR RETURN FOR MENU  
█
```

Figure B-4 Sample Directory

The number on the right of each program name is the program length in *sectors*. A sector is a length of measurement for the diskette in computer jargon. Underneath the list of program names is the number of unused sectors. A full diskette has 0 free sectors.

To get a diskette directory:

1. Type **DOS** and press **RETURN** to get the DOS utility program.
2. At the bottom of the screen display, the DOS utility asks:

SELECT ITEM

Enter:

A **RETURN**

3. The computer responds with:

DIRECTORY SEARCH SPEC, LIST FILE?

You can enter two responses:

1. To list the entire directory, press the **RETURN** key.
2. To *search* the directory for a specific file name, enter the file name. The computer displays the file name and its sector length, followed by the number of free sectors. If the file is not found, only the number of free sectors is displayed.

Anything entered other than a letter or file name results in no action or an ERROR-165 message.

Creating Your Own Disk Directory. Instead of leaving PILOT to enter the DOS utility to display a diskette directory, you can create your own directory.

Following is a program called MENU. If LOAded, SAVEd, and RUN as a PILOT program, MENU allows you to create your own directory. Number each entry so that you can select and automatically load a file by entering one number. Type in:

```
10 R: PILOT MENU SELECT PROGRAM
20 R: 28-OCT-80
30 R: HARRY B. STEWART
40 R:
50 C: @B1373 = 2           [ AUX1 = READ DIRECTORY.
60 C: $DIR = D:*. *       [ WILDCARD = ALL FILES.
70 C: #F = 1              [ FILE # = 1.
80 C: #R = 0              [ SCREEN ROW = TOP.
90 READ: $DIR,$FILE
100 R:
110 R: MAIN LOOP TO LIST FILENAMES.
120 R:
130 *LOOP
140 C: #C = 2              [ SET SCREEN COLUMN.
150 U: *FILELIST          [ LIST FILENAME.
160 T: \
170 READ: $DIR,$FILE
180 J(@B228 = 136): *DIRDONE [ EOF TEST.
190 C: #C = 20            [ SET SCREEN COLUMN.
200 U: *FILELIST          [ LIST FILENAME.
210 T:
220 C: #R = #R + 1        [ GO TO NEXT ROW.
230 READ: $DIR,$FILE
240 J(@B228 = 136): *DIRDONE
250 J(#R < 20): *LOOP
260 T: SCREEN ABOUT TO OVERFLOW — CONTINUE (Y/N)?\
270 A:                     [ GET ANSWER.
280 M: Y,                  [ WAS IT YES-LIKE?
290 JN: *DIRDONE          [ NO — ALL DONE.
300 C: #R = 0             [ YES — START AT TOP.
310 T: ↵                  [ CLEAR SCREEN. \
320 A: = $FILE           [ RESTORE ACCEPT BUFFER.
330 J: *LOOP
340 R:
350 R: DONE WITH DIRECTORY DISPLAY.
360 R:
370 *DIRDONE
380 CLOSE: $DIR
390 R:
400 R: FIND OUT WHICH FILE TO LOAD.
410 R:
420 *SELECT
430 T:
440 T: SELECTION? \
450 A: #N
460 J(#N < 1): *SELECT
470 U: *ITEMNAME
480 A: $FILENAME = D:$ITEM
490 M: $
500 JY: *SELECT
510 T: ↵ LOADING '$FILENAME'.
```

520 LOAD: \$FILENAME
 530 E:
 540 R:
 550 R: FORMAT FILENAME FOR PRINTING.
 560 R:
 570 *FILELIST
 580 C: \$LEFT =
 590 A: = \$FILE! [ANCHOR FILE SIZE.
 600 *FLOOP
 610 MS: 0!, 1!, 2!, 3!, 4!, 5!, 6!, 7!, 8!, 9!
 620 AY: = \$LEFT! [STRIP OFF # SECTORS.
 630 JY: *FLOOP
 640 POS: #C, #R [PRINT FILE NAME.
 650 T: \$LEFT\
 660 C: #C = #C + 13
 670 C: #N = #N + 1
 680 POS: #C, #R [PRINT FILE #
 690 T: (#N)\
 700 A: = \$LEFT [CONVERT NAME TO ...
 710 MS: → , [... I/O FORM.
 720 C: \$FILENAME = \$LEFT.\$RIGHT
 730 U: *ITEMNAME
 740 C: \$\$ITEM = \$FILENAME
 750 E:
 760 R:
 770 R: CREATE STRING ARRAY NAME.
 780 R:
 790 *ITEMNAME
 800 C: #X = 100 - #N [INVERT FOR SPEED.
 810 C: \$ITEM = Z#X['Z' KEEPS STRINGS ABOVE \$LEFT, \$MATCH & \$RIGHT
 FOR SPEED.
 820 E:

Returning to PILOT From the DOS Utility

To leave the DOS utility and return to PILOT, choose option B from the DOS directory, or press the **SYSTEM RESET** key.

When you return to the PILOT language, your screen displays:

ATARI PILOT (C) COPYRIGHT ATARI 1980

READY



and you are ready to go!

DON'T PANIC! (How To Handle Trouble With the Disk Drive)

There are three main types of problems that can occur while you are using your disk drive. These are described below with the suggested remedies. Keep in mind that switching the computer to OFF *clears its memory*.

Screen Displays "BOOT ERROR." Either you turned on the system without inserting a diskette, or the diskette was inserted upside down or backwards.

THE ATARI 410 PROGRAM RECORDER

Remedy: Turn off the power switch on the computer. **DO NOT TURN THE DISK DRIVE OFF FIRST.** Within 20 seconds, the BUSY light will go off. Open the disk drive door, insert the diskette properly, and turn it on again. If the BOOT ERROR message is displayed again, consult the section entitled "Boot Errors" in the *ATARI 400/800™ Disk Operating System Reference Manual*.

BUSY Light on Disk Drive Does Not Go Off. The BUSY light on the disk drive goes on when you start up or when the disk drive is *busy* sending information to or getting information from the computer. If the BUSY light remains on when the disk drive is not in use, something is wrong.

Remedy: See the remedy described for a BOOT ERROR problem.

The Computer Does Not Respond. This may have several causes.

Remedy: Turn the power switch on your computer to off and then to ON again. You may need to repeat this several times until you see the READY message on the screen.

The ATARI 410 Program Recorder stores programs on cassette tape. Ordinarily, one program is stored on each side of the cassette tape, however more than one program can be stored sequentially on each side if desired.

SPEAKING TO YOUR PROGRAM RECORDER

A program in the computer's memory can be *saved* on a cassette tape and *loaded* from the tape at a later time.

Saving a Program on a Cassette Tape

1. Make sure all cables are securely attached to their connectors. Turn on the computer.
2. Open the program recorder lid by pressing the STOP EJECT key.
3. Insert a cassette tape as shown in the *ATARI 410™ Program Recorder Operator's Manual*.
4. Use the REWIND or ADVANCE button to rewind the tape or move to the desired location on the tape counter.
5. If the tape has been rewound, set the tape counter (located below the lid) to 000 by pressing the tape counter button.

■ If you are saving more than one program on the tape, advance the tape a few counts past the end of the last program when saving a new program. Always write down the first and last counter numbers so you can find the starting point of each program. ■

5. Enter a program into the computer.
6. The SAVE command saves the program currently in memory on a cassette tape. **Programs saved on cassette cannot be given names.**

Enter:

SAVE C: and press **RETURN**

where: **C** specifies the cassette or program recorder.

Example:

SAVE C:

To save only a portion of a program, the program lines to be saved must be specified.

SAVE C: x,y

where: x is the first program line to be saved

y is the last program line to be saved

This allows you to save sections and modules out of an entire program.

Example:

SAVE C: 5,200 saves line 5 through 200

7. Upon pressing the **RETURN** key, you will hear two beeps. After the beeps, depress the **PLAY** and **RECORD** buttons at the same time. Then press the **RETURN** key again. At first, you will hear a high-pitched tone as blank leader is being saved. Then, as the program is being saved, you will hear a series of raspy tones coming from the television.
8. When the **READY** message and the cursor appear on the screen, the program is successfully saved. Write down the counter number that marks the end of the program.

This exercise takes you through the steps of saving a program onto cassette tape.

1. Turn on the computer with the program recorder connected.
2. Insert a new cassette tape. Rewind it to the beginning and set the tape counter to 000.
3. Type:
10 R: PRACTICE PROGRAM
20 T:
30 T: THIS PROGRAM LETS YOU PRACTICE
40 T: SAVING AND LOADING A PROGRAM
50 T: ON A CASSETTE TAPE.
60 E:
4. Type:
SAVE C: **RETURN**.
5. Press **PLAY** and **RECORD** at the same time.
6. Press **RETURN**.
7. When the **READY** message and the cursor appear on the screen, the program has been saved on the cassette tape. Write down the tape counter number.

Write-Protecting a Cassette Tape

To protect your programs on a cassette tape from being accidentally overwritten, you can *write-protect* your tape. Every cassette tape has two write-protect notches (shown in Figure B-1), one for each side of the tape. To write-protect one side of the tape, punch out the write-protect tab that is on the left when the side to be write-protected is facing up. Many prerecorded tapes are already write-protected for you.

If you decide to reuse a write-protected tape, place a piece of cellophane tape over the write-protect opening.

Loading a Program From a Cassette Tape

1. Make sure all cables and connectors are securely in place. Turn on the computer.
2. Insert the cassette tape containing the program you wish to load.
3. Rewind the cassette tape to the beginning by pressing the **REWIND** key. Set the tape counter to 000.
4. Fast forward the cassette tape to a position before the beginning of the program by pressing the **ADVANCE** key. Press the **STOP EJECT** key when the tape counter is close to the number marking the beginning of the program.
5. To load the program, type:

LOAD C: **RETURN**

where: **C** specifies the cassette, or program recorder

Examples:

LOAD C: loads a program from cassette tape into memory

6. Upon pressing the **RETURN** key, you will hear one beep. Press **PLAY** on the program recorder.
7. After about 5 seconds you will hear tones indicating that your program is being loaded. If the program recorder must search the tape for more than 9 seconds to find your program, it will give up looking.
8. When the screen displays the READY message, your program is successfully loaded.
9. LIST and RUN the program.

■ For additional help, read the first pages of the *ATARI 410™ Program Recorder Operator's Manual*. ■

The following exercise takes you through the steps of loading the PRACTICE program previously saved on tape.

1. Turn on the computer with the program recorder connected.
2. Insert the cassette containing the PRACTICE program.
3. Rewind the tape to the beginning. The tape counter should be set to 000.
4. Type:

LOAD C: **RETURN**

5. After you hear one beep, press **PLAY**, then press **RETURN**.
6. The computer makes tones as it loads the PRACTICE program. When a READY message appears on the screen, the program is loaded.
7. LIST and RUN the program.

Merging Programs

When loading a program from a cassette tape, you can *merge*, or combine, the program on tape with the current program in memory.

For more information about merging programs, refer to "Merging Programs" under "Loading Programs From a Diskette."

■ To avoid accidentally merging two programs, always type **NEW** **RETURN** before loading a program. ■

A Method To Manage Programs on Cassette Tapes

This section describes how to save up to 30 programs on one cassette **and** create a program directory.

This method centers on the program listed on the following pages. This program is designed to function as a directory within the cassette tape. It allows you to enter the program names saved on the tape and their corresponding counter numbers. Later, you can look up the program name and its counter number and advance the tape to its starting point.

■ This program must be entered as the first program on each tape. ■

Saving the Cassette Directory. To save the cassette directory, follow the steps below:

1. Enter the directory program into the computer. (This program contains its own "help system." None of the lines should be removed. Lines 200-400 will eventually be replaced with program names.)

```
10 R: CASSETTE PROGRAM GUIDE
15 T: ↑      [ESC SHIFT CLEAR]
20 T:
25 T:      ***CASSETTE PROGRAM GUIDE***
30 T:
35 T:
40 U: *HELP
99 *START
100 T: (PROGRAM NAME) (BEGIN) (END)
110 T:
190 R: START ADDING PROGRAMS AT 200
200 R: THERE ARE NOT YET ANY PROGRAMS
210 J: *AAAABAAAABAAAABAAAABAAAAB
220 J: *AAAABAAAABAAAABAAAABAAAAB
230 J: *AAAABAAAABAAAABAAAABAAAAB
240 J: *AAAABAAAABAAAABAAAABAAAAB
250 J: *AAAABAAAABAAAABAAAABAAAAB
260 J: *AAAABAAAABAAAABAAAABAAAAB
270 J: *AAAABAAAABAAAABAAAABAAAAB
280 J: *AAAABAAAABAAAABAAAABAAAAB
290 J: *AAAABAAAABAAAABAAAABAAAAB
300 J: *AAAABAAAABAAAABAAAABAAAAB
310 J: *AAAABAAAABAAAABAAAABAAAAB
320 J: *AAAABAAAABAAAABAAAABAAAAB
330 J: *AAAABAAAABAAAABAAAABAAAAB
340 J: *AAAABAAAABAAAABAAAABAAAAB
350 J: *AAAABAAAABAAAABAAAABAAAAB
```

360 J: *AAAABAAAABAAAABAAAABAAAAB
370 J: *AAAABAAAABAAAABAAAABAAAAB
380 J: *AAAABAAAABAAAABAAAABAAAAB
390 J: *AAAABAAAABAAAABAAAABAAAAB
400 J: *AAAABAAAABAAAABAAAABAAAAB
410 J: *AAAABAAAABAAAABAAAABAAAAB
420 J: *AAAABAAAABAAAABAAAABAAAAB
430 J: *AAAABAAAABAAAABAAAABAAAAB
440 J: *AAAABAAAABAAAABAAAABAAAAB
450 J: *AAAABAAAABAAAABAAAABAAAAB
460 J: *AAAABAAAABAAAABAAAABAAAAB
470 J: *AAAABAAAABAAAABAAAABAAAAB
480 J: *AAAABAAAABAAAABAAAABAAAAB
490 J: *AAAABAAAABAAAABAAAABAAAAB
990 *AAAABAAAABAAAABAAAABAAAAB
997 T:
998 T: CASSETTE TAPE SIZE IS: 673
999 E:
9000 *HELP
9010 T: DO YOU NEED HELP? \
9020 A:
9030 M: N
9040 U: *CLEAR
9050 EY:
9060 T: REWIND THE TAPE ALL THE WAY.
9070 T: SET THE TAPE COUNTER TO 000.
9080 T: LEAVE AT LEAST 10 COUNTS.
9090 T: BETWEEN THE END OF ONE PROGRAM
9100 T: AND THE BEGINNING OF THE NEXT.
9110 U: *RETURN
9120 EY:
9130 T: ** TO SAVE A NEW PROGRAM **
9140 T:
9150 T: SAVE IT TEMPORARILY ON THE
9160 T: REVERSE SIDE OF THE TAPE.
9170 T: THIS WILL GIVE YOU AN IDEA
9180 T: OF ITS SIZE. (IT'S A GOOD IDEA
9160 T: TO KEEP THIS SIDE OF THE TAPE EMPTY).
9200 T:
9210 T: NEXT, FLIP THE CASSETTE BACK
9220 T: TO THIS SIDE, TYPE: NEW,
9230 T: AND LOAD/RUN THIS PROGRAM
9240 T: TO FIND OUT WHERE TO SAVE
9250 T: YOUR PROGRAM.
9260 T:
9270 T: NEXT POSITION THE COUNTER
9280 T: AND SAVE YOUR COUNTER.
9290 T: INCLUDE THE BEGIN/END COUNT
9300 T: IN THIS PROGRAM.
9310 U: *RETURN
9320 EY:
9330 T: **RESAVING PROGRAMS**
9340 T:
9350 T: IT IS ONLY SAFE TO RESAVE
9360 T: PROGRAMS IF THEY GROW NO
9370 T: MORE THAN 10 LINES IN SIZE.

9380 T:
 9390 T: TO RESAVE A PROGRAM WHICH HAS
 9400 T: GROWN MUCH LARGER, SAVE IT
 9410 T: AS A NEW PROGRAM.
 9420 T: TO ERASE A PROGRAM, JUST
 9430 T: JUST REPLACE ITS NAME WITH:
 9440 T: (EMPTY)
 9450 T: DON'T CHANGE THE BEGIN/END
 9460 T: NUMBERS.
 9470 U: *RETURN
 9480 EY:
 9490 T:
 9500 T: WHEN FILLING AN (EMPTY) SPACE
 9510 T: WITH A PROGRAM, KEEP IN MIND
 9520 T: THAT, IN GENERAL:
 9530 T: EVERY 20 PROGRAM LINES
 9540 T: OCCUPY 12 COUNTS.
 9550 U: *RETURN
 9560 EY:
 9570 T:
 9580 E:
 9590 *RETURN
 9600 T:
 9610 T: (PRESS RETURN TO CONTINUE)\
 9620 A:
 9630 U: *CLEAR
 9640 T: MORE? \
 9650 A:
 9660 M: N
 9670 U: *CLEAR
 9680 E:
 9690 *CLEAR
 9700 T: ↑ [ESC SHIFT CLEAR]
 9710 E:

2. Insert a blank cassette tape into the program recorder. Rewind completely and set the tape counter to 000.
3. Advance the tape to about 010 on the tape counter. Remember this number.
4. Save the directory program by typing:

SAVE: C RETURN

5. On another tape, save the directory program again, repeating steps 1-4. This way you have the original, unchanged program to save onto other tapes.
6. LIST lines 200-250 of the directory to display the first lines to be filled with program names.
7. Enter the directory program name as the first entry in the directory.

Example:

200 T: CASSETTE GUIDE 010 071

(010 is the starting point of the directory. 071 is the ending point.)

8. Since you have changed the directory you must re-SAVE it. Rewind and position the tape at 010. ReSAVE the program by typing:

SAVE: C RETURN

■ The directory program is designed to remain the same size no matter how many programs you save! If you save the directory program starting at the same location on the tape every time, it will be resaved right over the old directory. ■

Adding Programs to the Directory and the Tape. To add a new program name to the directory and to the tape, follow these instructions:

1. LOAD the directory. RUN the directory to see the ending number of the last program saved.
2. Position the tape to the last program's ending number plus 10. If the last program ended at 090, position the tape to at least 100.
3. Type: **NEW** .
4. Enter the program to be saved.
5. Save the program:
Type **SAVE C:** .
6. Remember the beginning and ending points of the program.
7. Type **NEW** .
8. Rewind the tape and LOAD the program directory.
9. Type:

LIST 200,500

Replace the first available J: line after 200 with a T: command, followed by the program name and its beginning and ending tape numbers.

```
LIST 200,500
200 T: CASSETTE GUIDE 010 071
210 J: PROGRAM.1 080 100
220 J: *AAAABAAAABAAAABAAAABAAAAB
230 J: *AAAABAAAABAAAABAAAABAAAAB
240 J: *AAAABAAAABAAAABAAAABAAAAB
250 J: *AAAABAAAABAAAABAAAABAAAAB
.
.
.
490 J: *AAAABAAAABAAAABAAAABAAAAB
```

10. Since you have changed the directory again, you must reSAVE the directory by repeating steps 1-3 of "Saving the Cassette Directory."

THE ATARI PRINTERS

The ATARI printers type program statements or program outputs on paper, similar to a LIST or RUN on the screen. The **ATARI 820™ 40-Column Printer** uses standard adding machine paper. The **ATARI 822™ Thermal Printer** is a 40-column printer that uses heat-sensitive paper. The **ATARI 825™ 80-Column Printer** uses roll or fan-fold paper. The printer is a slow device compared to the computer, disk drive, or program recorder and may often seem to rest while printing. The printer is unable to print cursor control, escape, or graphic characters.

PRINTING A PROGRAM

1. Make sure the printer is attached to the computer and that both are turned on.
2. Enter the program to be printed on the computer.
3. To print the entire program, type:

SAVE P: `RETURN`

To print selected lines of a program, type:

SAVE P: X,Y

where: **P:** specifies the printer

x is the first line of the program to be saved

y is the last line of the program to be saved

4. When you press the `RETURN` key, the printer starts printing the program on the paper.
5. When the program is printed, the READY message is displayed on the screen.
6. Press the yellow button or move the roller to advance the paper.

The following exercise takes you through the steps of printing a program:

1. Turn on the ATARI 400/800 computer and the printer, and the **ATARI 830™ Interface Module** if you are using the ATARI 825 Printer.
2. Enter or load the PRACTICE program.
3. Type:
SAVE P: `RETURN`
4. When the program has been printed, advance the paper and tear off the program.

APPENDIX C

SCREEN EDITING

CONCEPTS INTRODUCED

- The Screen Editor
- The Screen Editing Keys
 - The RETURN Key
 - The SHIFT Key
 - The CAPS/LOWER Key
 - The TAB Key
 - TAB Set
 - TAB Clear
 - TAB
 - The CTRL Key
 - The Inverse Video Key
 - The CLEAR Key
- The Cursor Control Keys
 - Cursor Up ↑
 - Cursor Down ↓
 - Cursor Right →
 - Cursor Left ←
- Editing the Current Display Line
 - Backspacing
 - Deleting
 - Character Deletion
 - Line Deletion
 - Inserting
 - Character Insertion
 - Line Insertion
- Deferred Cursor Movements

THE SCREEN EDITOR

Anything displayed on the television screen may be edited and changed in immediate mode, using the built-in screen editor. The screen editor is part of the computer (you can't see it). It lets you double-check and correct your input *before* it goes into the computer's memory. By learning to use the screen editor, you can correct problems anywhere in your input.

THE SCREEN EDITING KEYS

The black keys shown on the diagram below control the typewriter-like functions of the keyboard.

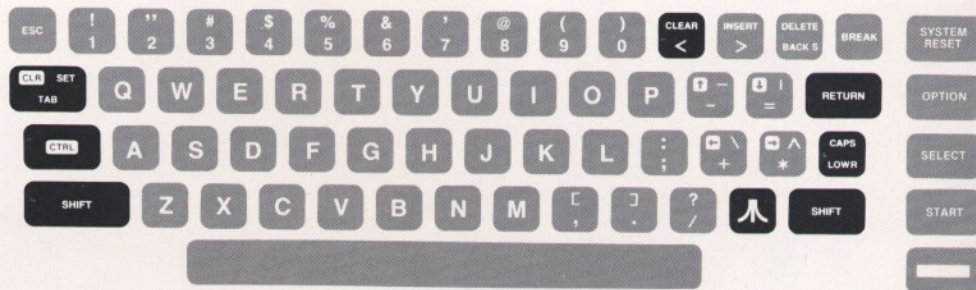


Figure C-1 The ATARI Keyboard: Screen Editing Keys

THE RETURN KEY

The **RETURN** key has three functions:

1. It moves the cursor to the left margin and down one line of the screen.
2. It marks the end of an input line.
3. It activates the computer to respond.

The function of the **RETURN** key depends upon what the computer is doing at the time the **RETURN** key is pressed.

THE SHIFT KEY

The **SHIFT** key, when pressed simultaneously with another key, accesses the character or function displayed on the top half of a key. The **SHIFT** key is located on both ends of the bottom row of the keyboard.

THE CAPS/LOWER KEY

The **CAPS LOWER** key accesses either lowercase or uppercase alphabetic letters. If the key is pressed prior to typing any letters, the letters are lowercase. If the key is pressed while holding down the **SHIFT** key, uppercase letters are displayed instead.

THE TAB KEY

The **CLR SET TAB** key operates like the **TAB** key on a regular typewriter to set or clear automatic stopping positions of the cursor.

TAB Set

To set a tab position, move the cursor to the desired stopping position and press the **TAB** key while holding down the **SHIFT** key. Up to 38 tabs may be set across the screen.

TAB Clear

To clear a tab position, move the cursor to a TAB set position. Hold down the **CTRL** key (located directly below the **TAB** key) while pressing the **TAB** key.

TAB

The **TAB** key by itself moves the cursor over to the next tab position. If the next tab position is on the following line, the cursor drops down to the first tab position of the following line.

THE CTRL KEY

The CONTROL key (labeled **CTRL** on the keyboard), functions as a second type of **SHIFT** key. The **CTRL** key accesses the graphics character set and the cursor control functions. Holding down the **CTRL** key while pressing an alphabetic key causes a graphics character from the graphic character set to appear on the screen. If **CTRL** is pressed with a cursor control key, it moves the cursor.

THE INVERSE VIDEO KEY

The **^** key displays any following characters in inverse video (shown as a blue character within a white square). To return to the normal display, press the **^** key again.



Figure C-2 Using the Inverse Video Key

THE CLEAR KEY

The **CLEAR** key is used to clear the screen. "Clearing the screen" means to erase all characters from the screen (but not from memory) and position the cursor in the upper-left corner (the "home" position).

To clear the screen, hold down the **SHIFT** key or the **CTRL** key while pressing the **CLEAR** key.

THE CURSOR CONTROL KEYS

The black keys shown on the diagram below show the cursor control keys.

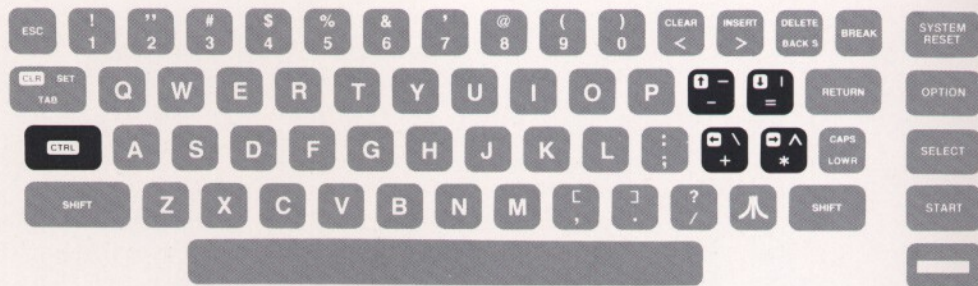


Figure C-3 The ATARI Keyboard: The Cursor Control Keys

The cursor control functions, **CTRL** ↑, **CTRL** ↓, **CTRL** →, and **CTRL** ← move the cursor in the direction shown by the arrows on the keytops. The **CTRL** key MUST be held down while pressing the cursor control keys. When the cursor moves over a character, that character is displayed in "inverse video."

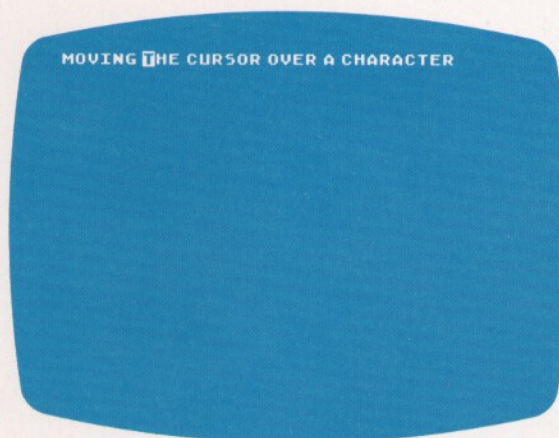




Figure C-4 Moving the Cursor Over a Character

The character remains unchanged unless you type another character on top of it.

CURSOR UP

Pressing the  key while holding down the  key moves the cursor up one line.

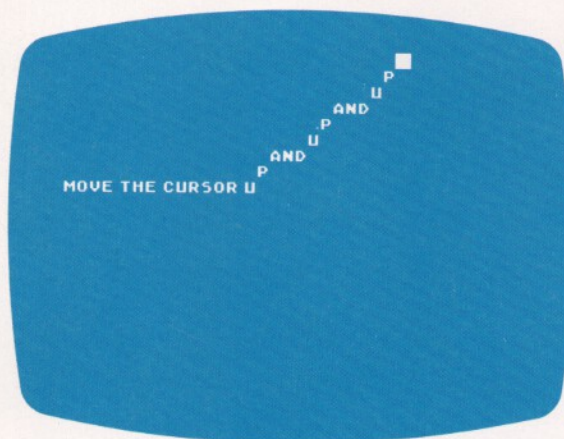


Figure C-5 Using the Cursor Up Key

If the cursor reaches the top of the screen, it "wraps around" to the bottom in the same column.

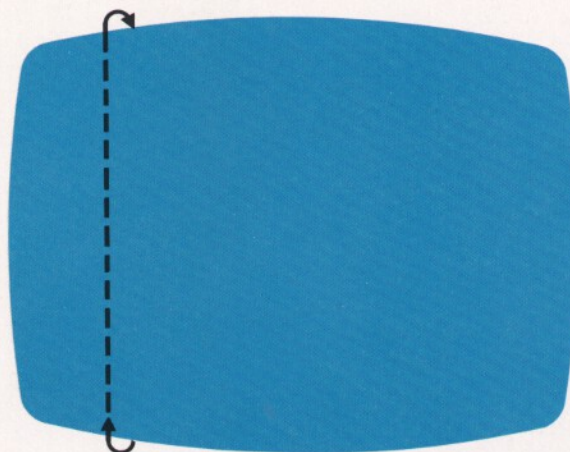




Figure C-6 Cursor Up Wraparound

CURSOR DOWN

Pressing the  key while holding down the  key moves the cursor down one line.

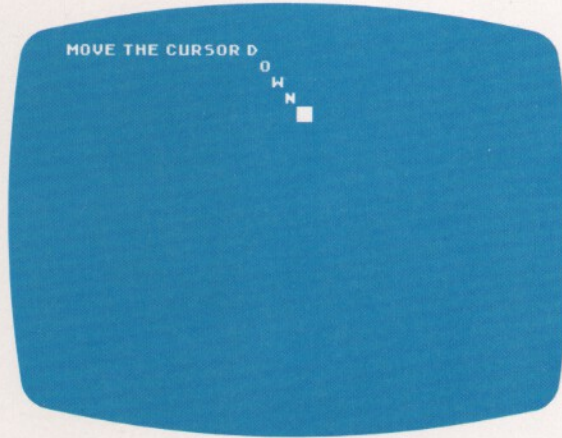


Figure C-7 Using the Cursor Down Key

If the cursor reaches the bottom of the screen, it wraps around to the top in the same column.

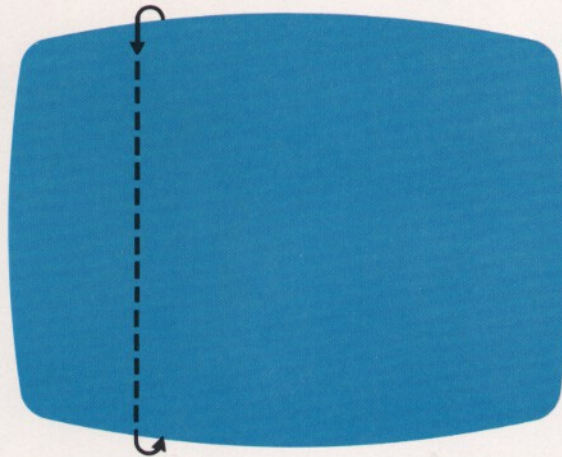




Figure C-8 Cursor Down Wraparound

CURSOR RIGHT

Pressing the  key while holding down the  key moves the cursor one space to the right.

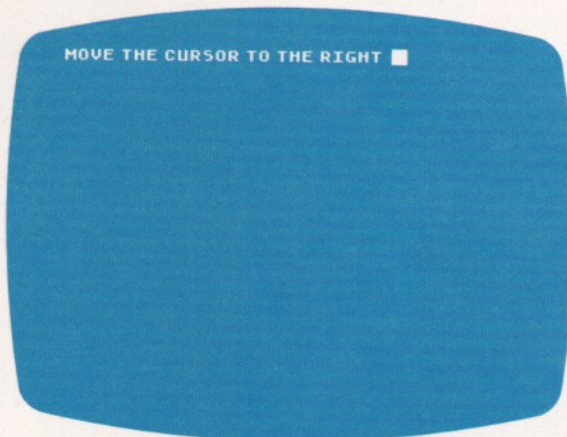
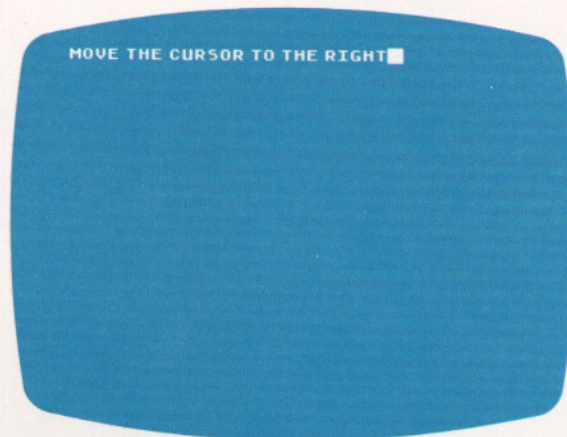


Figure C-9 Using the Cursor Right Key

If the cursor reaches the right side of the screen, it wraps around to the left side in the same row.

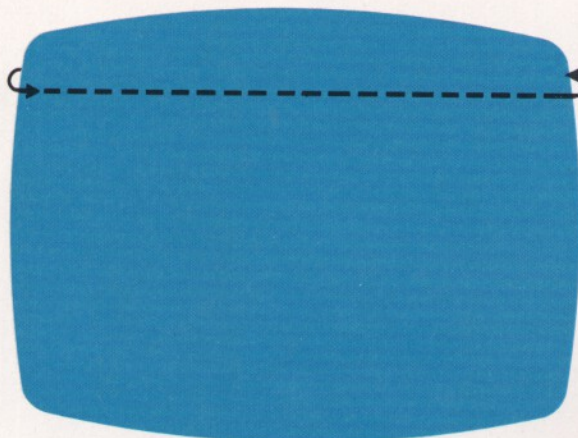




Figure C-10 Cursor Right Wraparound

CURSOR LEFT

Pressing the  key while holding down the  key moves the cursor to the left one space.

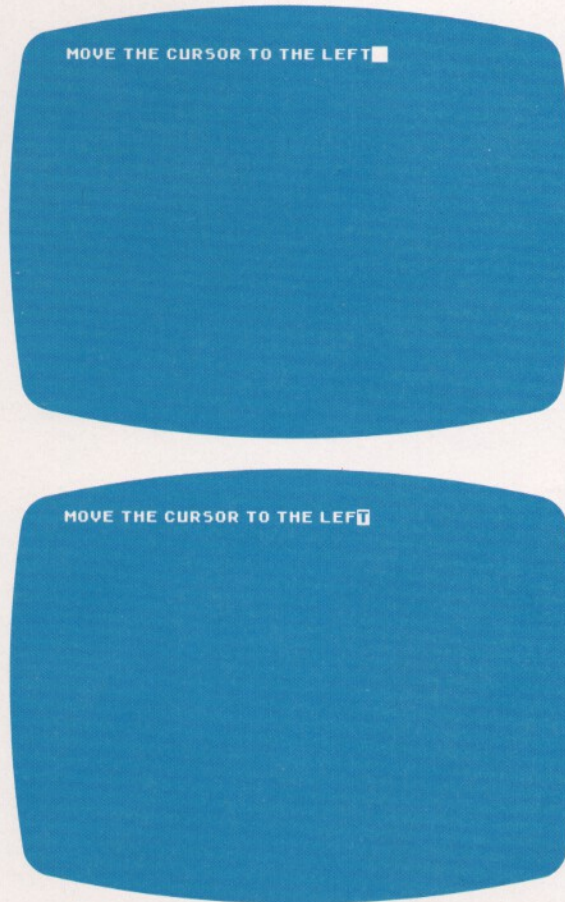


Figure C-11 Using the Cursor Left Key

If the cursor reaches the left side of the screen, it wraps around to the right side of the same row.

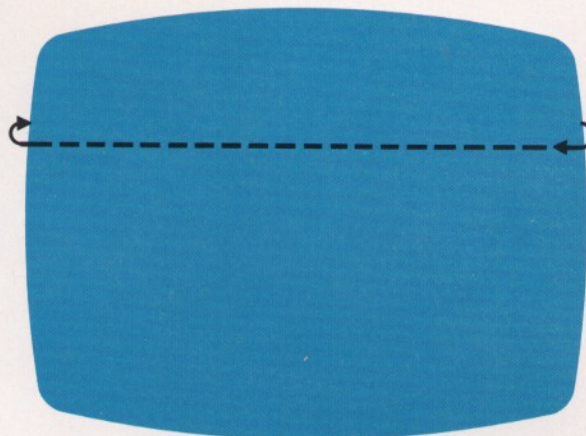


Figure C-12 Cursor Left Wraparound

EDITING THE CURRENT DISPLAY LINE

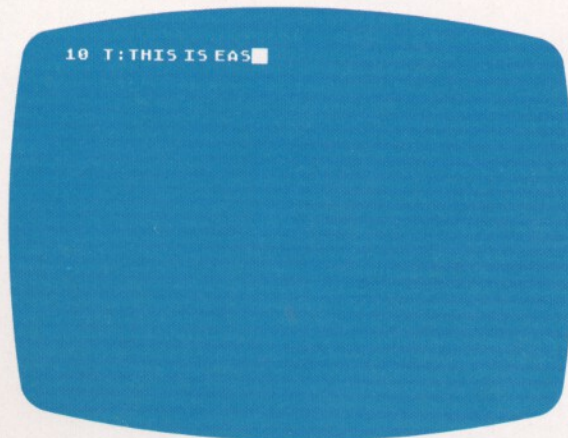
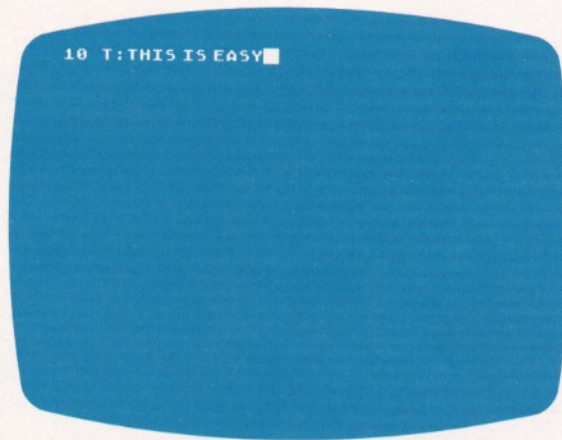
Backspacing

The **DELETE BACK S** key moves the cursor back one space at a time. Backspacing the cursor erases any character it moves over. The **DELETE BACK S** key is located on the top row of the keyboard.

Enter the following line (do not press the **RETURN** key):

10 T:THIS IS EASY#

To erase the EASY, backspace the cursor to the space following the **IS** by pressing the **DELETE BACK S** key five times.



10 T:THIS IS EA■

10 T:THIS IS E■

10 T:THIS IS ■

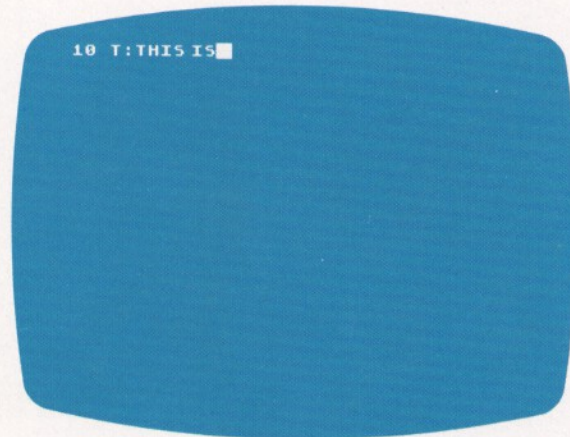


Figure C-13 Backspacing the Cursor Using the DELETE BACK S Key

Deleting

Character Deletion. Pressing the `DELETE BACK S` key while holding the `CTRL` key erases the character *under* the cursor. The characters to the right of the cursor move one space to the left, making the line shorter.

Enter the following line (do not press the `RETURN` key):

```
10 T:THIS IS NOT EASY#
```

Using the `CTRL` and `←` key move the cursor on top of the N in NOT.



Figure C-14 Single-Character Deletion

We want to delete the characters NOT. To delete the N press the `CTRL` and `DELETE BACK S` key once.

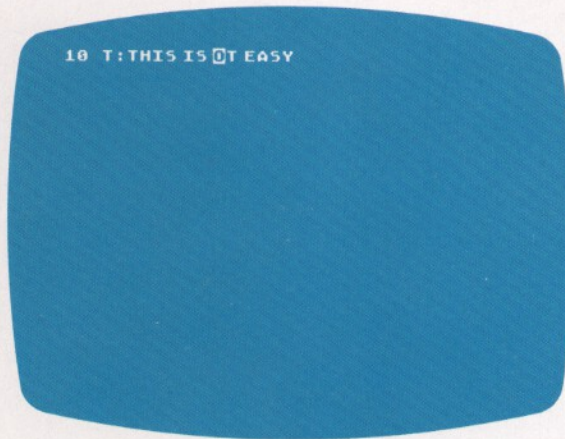


Figure C-15 Multiple-Character Deletion

Great! The N is deleted and the characters OT EASY move one space to the left.

To delete the OT and a space, hold down the **CTRL** key and press the **DELETE BACK S** key three times.

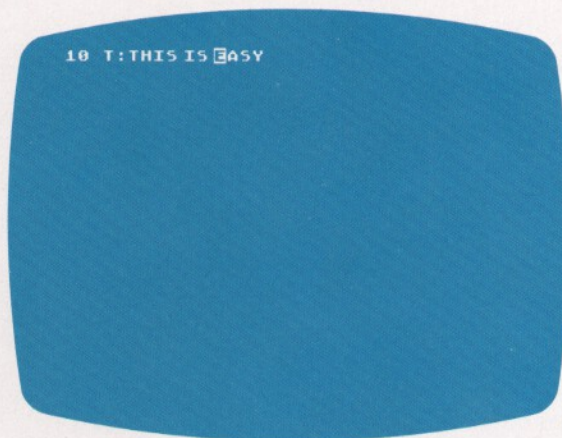


Figure C-16 Character Deletion Completed

Wasn't that easy? To exit the line, press **RETURN**.

Line Deletion. The line deletion function erases one line from the screen, but *not from memory*. To delete a line, move the cursor to any space on that line. Hold down the **SHIFT** key and press the **DELETE BACK S** key once. The line disappears, moving all the lines below the deleted line up one line. This leaves a blank line at the bottom of the screen.

Enter the following lines:

```
10 T: SCREEN EDITING IS EASY
20 T: ON MY ATARI COMPUTER.
30 T: DON'T YOU AGREE?
```

Let's delete line 20 from the screen *but not from memory*. Move the cursor up to line 20:

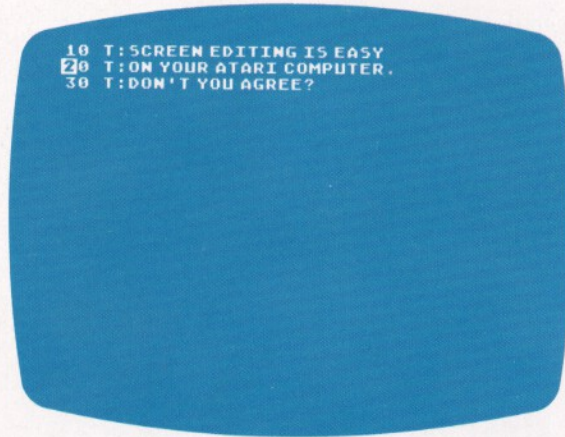


Figure C-17 Line Deletion: Positioning the Cursor

Press the **SHIFT** and **DELETE BACK S** key once. The screen displays:

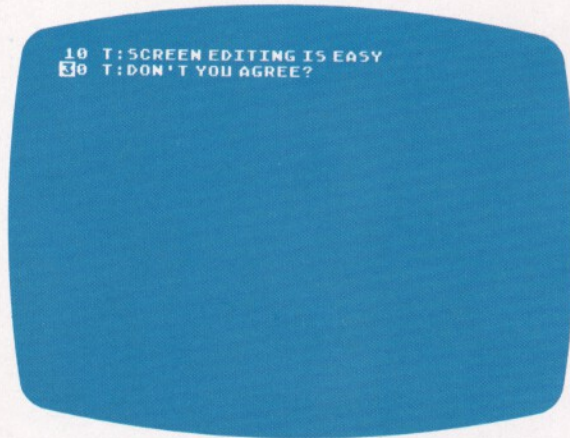


Figure C-18 Line Deletion Completed

It's that easy! Line 20 is erased from the screen. Now, LIST the program for a surprise!

```
10 T: SCREEN EDITING IS EASY
30 T: DON'T YOU AGREE?
LIST
10 T: SCREEN EDITING IS EASY
20 T: ON MY ATARI COMPUTER.
30 T: DON'T YOU AGREE?

READY
■
```

Line 20 is still in memory!

Inserting

Character Insertion. Pressing the **INSERT** key while holding the **CTRL** key inserts a space beneath the cursor. The cursor remains on the space so that a new character may be entered. All characters to the right of the cursor move one space to the right, making the line longer.

Enter the following line:

10 T: THIS IS EASY

Using the **CTRL** and **←** keys, move the cursor on top of the E in EASY.

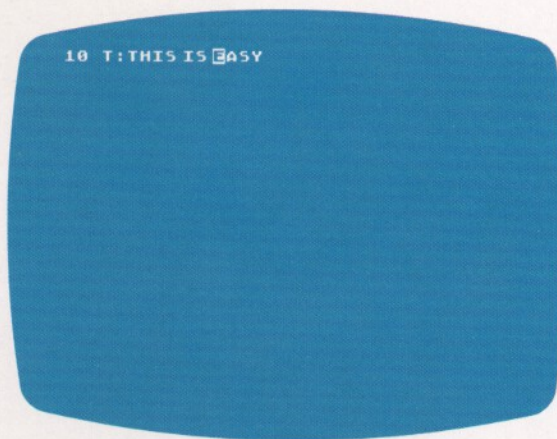


Figure C-19 Character Insertion: Positioning the Cursor

Let's insert the characters **VERY** between the **IS** and **EASY** to make the message read **THIS IS VERY EASY**. To insert enough space between the **IS** and **EASY** to fit **VERY**, press the **CTRL** and **INSERT** keys five times.

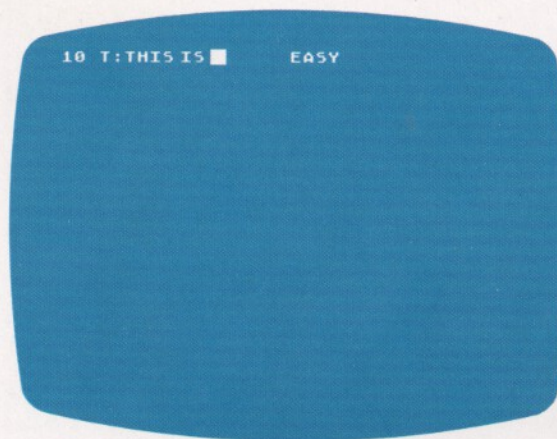


Figure C-20 Character Insertion: Creating a Space

Now, enter **VERY** and press the **RETURN** key. The line should now read:

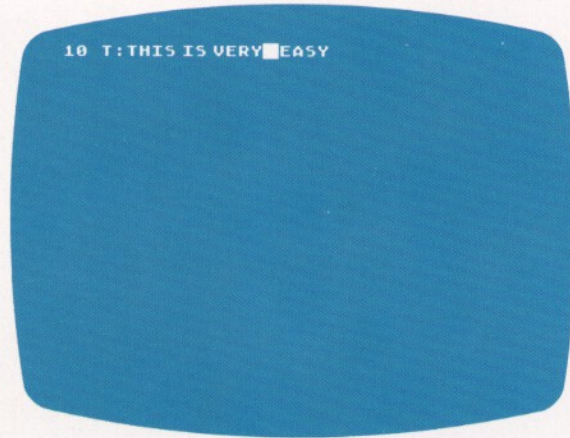


Figure C-21 Character Insertion Completed

Line Insertion. The line insertion function inserts a line onto the screen at the current cursor position. To insert a line, move the cursor to where you want to insert the line. Hold down the **SHIFT** key and press the **INSERT** key once. All the lines below the cursor move down one line, leaving a blank line at the current cursor position.

■ Any information on the bottom line of the screen will disappear. ■

Enter the following lines:

```
10 T: SCREEN EDITING IS EASY
20 T: DON'T YOU AGREE?
```

Let's insert a line between lines 10 and 20. To insert a line, move the cursor up to line number 20.

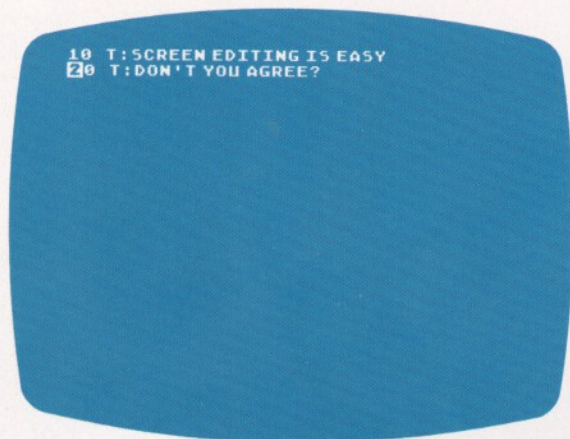


Figure C-22 Line Insertion: Positioning the Cursor

Hold down the **SHIFT** key and press the **INSERT** key. Line 20 drops down one line.

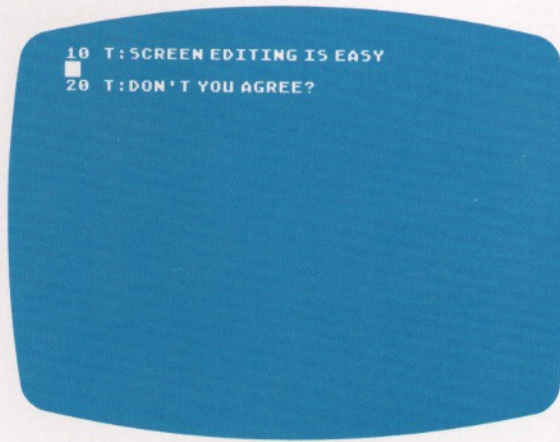


Figure C-23 Line Insertion: Creating a Space

Now there is room to enter a new program statement right between the other two:

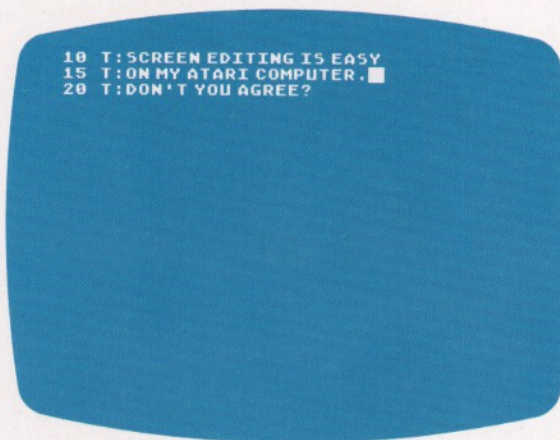


Figure C-24 Line Insertion Completed

That was easy, wasn't it?

DEFERRED CURSOR MOVEMENT

Cursor movements may be placed within program statements to be executed only upon the RUN command. To program cursor movements, press the **ESC** key prior to pressing the **CTRL** and cursor control keys. On the screen an insert arrow appears where you want the cursor movement. A program statement with a deferred clear screen movement (**ESC CTRL SHIFT CLEAR**) looks like this:

```
10 T:
```

The following program demonstrates how deferred cursor movements work. Type it in and give it a RUN:

```
10 R: CHARACTER SPIRAL
20 *BEGIN
25 C: #A = 0
30 T: ↵ [ CLEAR SCREEN ]
40 T: THIS PROGRAM WILL DRAW A SPIRAL
50 T: USING ANY CHARACTER YOU CHOOSE.
60 T:
70 T: WHAT CHARACTER WOULD YOU LIKE
80 T: TO USE? \
90 A: $CH
100 PA: 30
110 T: ↵ [ CLEAR SCREEN ]
115 *ERASE
120 POS: 19,11
130 U: *CHAR
140 C: #S = 1
150 *START
160 C: #C = 0
170 *TOP
180 U: *RIGHT
190 U: *CHAR
200 C: #C = #C + 1
210 J(#C#S): *TOP
220 C: #C = 0
230 *RIGHTSIDE
240 U: *DOWN
250 U: *CHAR
260 C: #C = #C + 1
270 J(#C#S): *RIGHTSIDE
280 C: #S = #S + 1
290 C: #C = 0
300 *BOTTOM
310 U: *LEFT
320 U: *CHAR
330 C: #C = #C + 1
340 J(#C<#S): *BOTTOM
350 C: #C = 0
360 *LEFTSIDE
370 U: *UP
380 U: *CHAR
390 C: #C = #C + 1
400 J(#C<#S): *LEFTSIDE
410 C: #S = #S + 1
420 J(#S<24): *START
430 C: #A = (#A + 1)\2
432 R: [ ESC, CNTRL, RT. ARROW, ESC, CNTRL, LFT ARROW
435 C: $CH = →←
440 J(#A = 1): *ERASE
445 PA: 360
446 J: *BEGIN
450 *RIGHT
455 R: [ ESC, CNTRL, RT. ARROW ]
460 T: → \
470 E:
```



```

480 *DOWN
485 R: [ ESC, CNTRL, DOWN ARROW ]
490 T: ↓ \
500 E:
510 *LEFT
515 R: [ ESC, CNTRL, LFT ARROW ]
520 T: ← \
530 E:
540 *UP
545 R: [ ESC, CNTRL, UP ARROW ]
550 T: ↑ \
560 E:
570 *CHAR
580 T: $CH \
585 R: [ ESC, CNTRL, LFT ARROW ]
590 T: ← \
600 E:

```

THIS PROGRAM WILL DRAW A SPIRAL
USING ANY CHARACTER YOU CHOOSE.
WHAT CHARACTER WOULD YOU LIKE
TO USE? ?■

```

????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????
????????????????????????????????

```

Figure C-25 Character Spiral Program RUN

APPENDIX D

CONTROLLERS

CONCEPTS INTRODUCED

Sensing
Sensing the Position of the Joystick
Sensing the Joystick's Trigger
Sensing the Position of the Paddle
Sensing the Paddle Trigger

The ATARI joysticks and paddles are called *controllers*. Controllers send information to the computer. To use this information, a program must be able to *sense* it.

SENSING

A controller sends its position to the computer. A joystick has 8 positions and the paddle has 227 positions. When a program *senses* the information from a controller, it receives a *value corresponding to the position of the controller*. This information is stored in read-only sensing variables. (Read-only variables can only be read; their values cannot be changed by the user.)

The value of a sensing variable is a number. A sensing variable is usable anywhere a numeric variable can be used.

SENSING THE POSITION OF THE JOYSTICK

Up to 4 joysticks can be attached at one time. There are 8 sensing variables in ATARI PILOT; 4 for the joysticks and 4 for the joystick's triggers.

They are:

4 joystick sensing variables:

%J0, %J1, %J2, %J3

4 joystick trigger sensing variables:

%T8, %T9, %T10, %T11

The value of %J0, %J1, %J2, and %J3 depends upon the position of the joystick, as shown below:

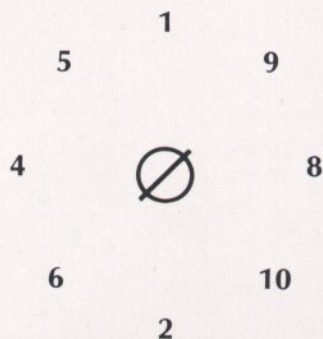


Figure D-1 Joystick Positions

This is how it works. Below are four diagrams, each representing a joystick. In each diagram is an arrow showing the direction the joystick is pointing. Enclosed in a box beneath the diagram is the value of that joystick's sensing variable.

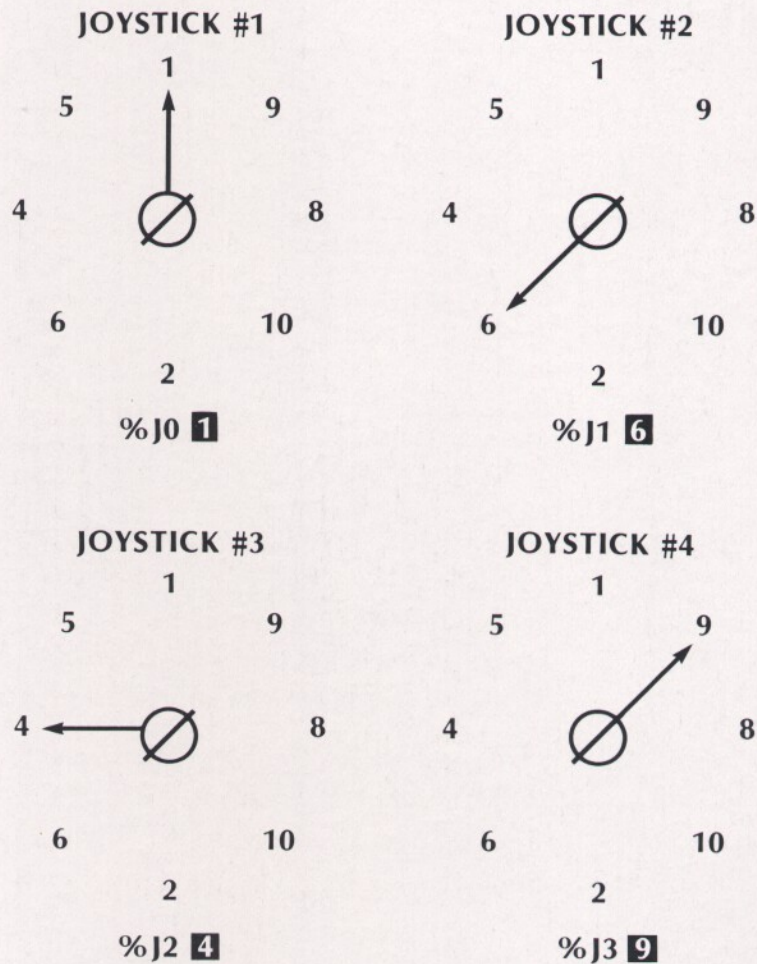


Figure D-2 Sample Joystick Sensing

Following is a short program that demonstrates how the joystick sensing variable works:

Type in:

```
10 R: JOYSTICK SENSING
20 *START
30 T: JOYSTICK #1 POSITION = %J0
40 PA: 60
50 J: *START
```

Plug the joystick into controller jack #1.

RUN the program. Move the control handle around. As it moves you should see its position value displayed every second. To sense the joystick more frequently, change the PAUSE command (PA:).

SENSING THE JOYSTICK'S TRIGGER

The value of the joystick trigger sensing variables, %T8, %T9, %T10, and %T11 depends upon whether or not the trigger is pressed:

0 if trigger is up

1 if trigger is pressed

If pressure on the trigger is not sensed within 5/60 of a second after it was last pressed, the trigger variable value becomes 0.

Below is an example of how the four trigger sensing variables work. The value in the box is the sensing variable's value:

JOYSTICK #1 TRIGGER	JOYSTICK #2 TRIGGER	JOYSTICK #3 TRIGGER	JOYSTICK #4 TRIGGER
pressed	not pressed	not pressed	pressed
% T8 1	% T9 0	% T10 0	% T11 1

Figure D-3 Sample Joystick Trigger Sensing

Example: Sensing a joystick trigger

```
100 J(%T8): *SHOOT
```

```
.
```

```
.
```

```
200 *SHOOT
```

(Branches to *SHOOT if the joystick #1 trigger is pressed.)

Following is a short program that demonstrates how the joystick trigger sensing variables work. It is the same program used to sense the joystick variables except for the changes in lines 10 and 30.

```
10 R: JOYSTICK TRIGGER SENSING
20 *START
30 T: JOYSTICK #1 TRIGGER = %T8
40 PA: 60
50 J: *START
```

SENSING THE POSITION OF THE PADDLE

Up to 4 pairs of paddle controllers may be connected to the computer at once; making a total of 16 paddle sensing variables: 8 for the paddles' positions, and 8 for the paddles' triggers.

8 paddle-sensing variables:

%P0, %P1, %P2, %P3, %P4, %P5, %P6, %P7

8 paddle/trigger-sensing variables:

%T0, %T1, %T2, %T3, %T4, %T5, %T6, %T7

The value of %P0 through %P7 depends upon the position of the paddle:

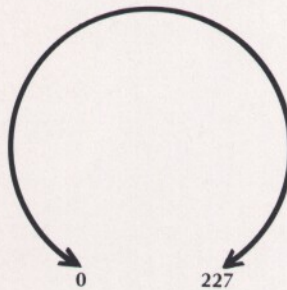


Figure D-4 Paddle Positions

The paddle positions range from 0 to 227 in a full clockwise rotation.

Below are two diagrams representing two paddles' positions:

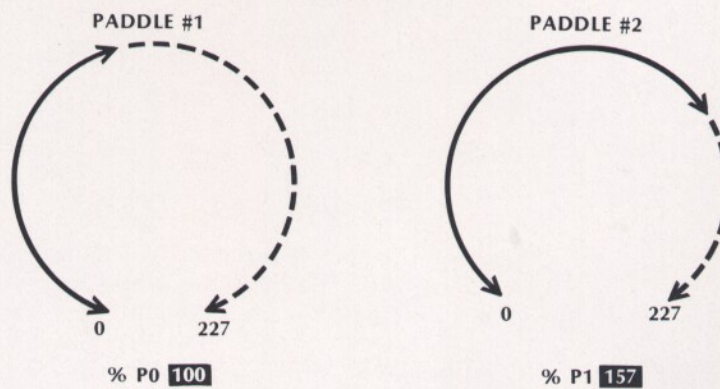
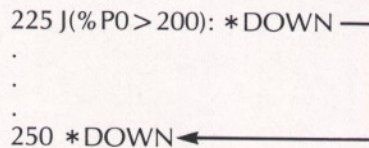


Figure D-5 Sample Paddle Positions

Example: Sensing the Paddle Position

```
225 J(%P0 > 200): *DOWN
.
.
.
250 *DOWN ←
```



(Branches to *DOWN if paddle #1 is turned nearly all the way clockwise.)

To see how %P0 through %P7 work, type in this program:

```
10 R: PADDLE SENSING
20 T: *START
30 T: PADDLE #1 POSITION = %P0
40 PA: 60
50 J: *START
```

RUN the program, turning the paddle clockwise and counterclockwise.

SENSING THE PADDLE TRIGGER

The paddle triggers, like the joystick triggers, have two values consisting of:

- 0 if trigger is up
- 1 if trigger is depressed

To create a Paddle Trigger Sensing Program, use the Paddle Sensing Program and replace lines 10 and 30 with the following:

```
10 R: PADDLE TRIGGER SENSING
30 T: PADDLE #1 TRIGGER = %T0
```

The following program does a "paddle sketch," using the graphics screen and the paddle controller sensing values. Type it in and give it a RUN:

```
10 R: ***ATARI PADDLE-SKETCH ***
20 R:
30 T: Plug the paddle controllers into controller jack #1.
40 T: Use the paddle on the right.
50 T:
60 T: To draw: turn the paddle.
70 T: To pause: press the trigger.
80 T: To resume: press it again.
90 T: To stop: press BREAK.
100 T:
110 T: (press RETURN to continue)\
120 A:
130 R: BEGIN PROGRAM
200 *LOOP
210 , U(%T0): *PAUSE
220 , PA: 15
230 , GR: DRAW 1
240 , R: change direction
250 , GR: TURNTO (36 * %P0 / 22)
260 , J: *LOOP
300 *PAUSE
310 , PA: 10
320 , E(%T0); █ resume on trigger
330 , J: *PAUSE
```

APPENDIX E

PILOT I/O ERROR CODES

- 130 A nonexistent device was specified.
- 131 A READ command followed a WRITE command with the same device specified.
- 135 A WRITE command followed a READ command with the same device specified.
- 136 End of file condition.
- 138 Device timeout; device doesn't respond. (See Note)
- 139 Device NAK. (See Note)
- 140 Serial bus framing error. (See Note)
- 141 Screen cursor out of range (READ from or WRITE to 'S').
- 142 Serial bus data frame overrun. (See Note)
- 144 Device DONE error. (See Note)
- 145 Disk read after write compare error. (See Note)
- 146 Function not implemented for device (e.g., OUT: K)
- 147 Insufficient RAM for operating the graphics screen.
- 160 Disk drive number error.
- 161 Too many concurrent disk files being accessed.
- 162 Disk is full (no free sectors).
- 163 Fatal system data I/O error.
- 164 File number mismatch. (See Note)
- 165 Disk file naming error.
- 167 Disk file locked.
- 169 Disk directory full (64 files).
- 170 Disk file not found in directory.

Note: These errors indicate problems over which the user has no direct control; they are due to hardware problems and should seldom be seen.

1027 PRINTER
1050 DISC DRIVE
800/538-8547

**LIMITED 90-DAY WARRANTY
ON ATARI® PERSONAL COMPUTER PRODUCTS**

ATARI, INC. ("ATARI") warrants to the original consumer purchaser that this ATARI Personal Computer Product (not including computer programs) shall be free from any defects in material or workmanship for a period of 90 days from the date of purchase. If any such defect is discovered within the warranty period, ATARI's sole obligation will be to repair or replace, at its election, the Computer Product free of charge on receipt of the unit (charges prepaid, if mailed or shipped) with proof of date of purchase satisfactory to ATARI at any authorized ATARI Service Center. For the location of an authorized ATARI Service Center nearest you, call toll-free:

In California (800) 672-1430
Continental U.S. (800) 538-8547

or write to: Atari, Inc.
Customer Service Department
1340 Bordeaux Drive
Sunnyvale, CA 94086

YOU MUST RETURN DEFECTIVE COMPUTER PRODUCTS TO AN AUTHORIZED ATARI SERVICE CENTER FOR IN-WARRANTY REPAIR.

This warranty shall not apply if the Computer Product: (i) has been misused or shows signs of excessive wear, (ii) has been damaged by being used with any products not supplied by ATARI, or (iii) has been damaged by being serviced or modified by anyone other than an authorized ATARI Service Center.

ANY APPLICABLE IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE HEREBY LIMITED TO NINETY DAYS FROM THE DATE OF PURCHASE. CONSEQUENTIAL OR INCIDENTAL DAMAGES RESULTING FROM A BREACH OF ANY APPLICABLE EXPRESS OR IMPLIED WARRANTIES ARE HEREBY EXCLUDED. Some states do not allow limitations on how long an implied warranty lasts or do not allow the exclusion or limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

**DISCLAIMER OF WARRANTY
ON ATARI COMPUTER PROGRAMS**

All ATARI computer programs are distributed on an "as is" basis without warranty of any kind. The entire risk as to the quality and performance of such programs is with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

ATARI shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss, or damage caused directly or indirectly by computer programs sold by ATARI. This disclaimer includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

REPAIR SERVICE

If your ATARI Personal Computer Product requires repair other than under warranty, please contact your local authorized ATARI Service Center for repair information.

IMPORTANT: If you ship your ATARI Personal Computer Product, package it securely and ship it, charges prepaid and insured, by parcel post or United Parcel Service.